Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Институт радиоэлектроники и информационных технологий — РТФ

Н. В. ПАПУЛОВСКАЯ

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ПРОГРАММИРОВАНИЯ ТРЕХМЕРНОЙ ГРАФИКИ

Учебно-методическое пособие

Министерство образования и науки Российской Федерации
Уральский федеральный университет
имени первого президента России Б. Н. Ельцина

Н. В. Папуловская

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ПРОГРАММИРОВАНИЯ ТРЕХМЕРНОЙ ГРАФИКИ

Рекомендовано методическим советом
Уральского федерального университета
в качестве *учебно-методического пособия*
для студентов, обучающихся по направлению подготовки
09.03.01 «Информатика и вычислительная техника»

Екатеринбург
Издательство Уральского университета
2016

УДК 004.421.2:004.92-023.5(075.8)
ББК 32.973я73+32.972.131.1я73
П17

Рецензенты:
Урал. ин-т экономики, управления и права (завкаф. математики и естественно-научных дисциплин, доц., канд. физ.-мат. наук *С. П. Трофимов*);
завкаф., доц., канд. техн. наук *С. В. Фёдорова* (НЧОУ ВО «Технический университет УГМК»)

Научный редактор — проф., д-р техн. наук *Л. Г. Доросинский*

На обложке — иллюстрация Андрея Кобелева

Папуловская, Н. В.
П17 Математические основы программирования трехмерной графики : учебно-методической пособие / Н. В. Папуловская. — Екатеринбург : Изд-во Урал. ун-та, 2016. — 112 с.
ISBN 978-5-7996-1942-8

В учебном издании рассмотрены алгоритмические основы построения и анимации трехмерных изображений. Изучается программный интерфейс, связывающий прикладную программу и систему компьютерной графики. Изложены основные математические и программные принципы создания трехмерных объектов с использованием графической библиотеки OpenGL. Сформулированы цели лабораторных работ, изложена необходимая информация для разработки графических программ визуализации анимационных трехмерных сцен.

Учебно-методическое пособие предназначено для бакалавров.

Библиогр.: 11 назв. Табл. 2. Рис. 43.

УДК 004.421.2:004.92-023.5(075.8)
ББК 32.973я73+32.972.131.1я73

ISBN 978-5-7996-1942-8

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Данное учебное пособие представляет собой вводный курс программирования компьютерной графики. Он включает описание основных понятий формирования трехмерных объектов и сцен. Изучается программный интерфейс, позволяющий осуществлять взаимодействие прикладной программе с оборудованием современного компьютера. Методические указания к лабораторному практикуму являются вспомогательным материалом для изучения технологии программирования трехмерной графики и содержат справочную информацию по графической библиотеке OpenGL и вспомогательным библиотекам.

Пособие состоит из трех разделов. В *разделе А* представлен математический аппарат, используемый при построении компьютерной графики. В первой главе вводятся основные математические понятия. Данный математический аппарат заимствован из векторного анализа и линейной алгебры. Однако в компьютерной графике его принято излагать иначе. Если в курсе математики студенты изучают сначала линейную алгебру, а затем знакомятся с векторными пространствами, то в математике для построения графики все наоборот. Сначала будет рассмотрено представление объектов в математических пространствах, которое приводит к использованию линейной алгебры. Использование подхода описания объектов, независящих от координат, вызвано желанием показать, что все базовые концепции геометрических объектов не зависят от способа преобразований. Такой подход используется в объектно ориентированном программировании, где работа осуществляется над объектами, а не над способами их представления.

Вторая глава посвящена проецированию и описанию стандартных проективных преобразований. В графических системах организация проективных преобразований отдается прикладной программе. Конечно, современные графические API избавляют прикладного программиста от забот, связанных с созданием собственных средств выполнения проективных преобразований. Однако знание основ этого процесса помогает понять, как в графической системе работает конвейер обработки данных, базирующийся на операциях с 4×4 матрицами.

В третьей главе пособия рассматривается физика взаимодействия света с поверхностями объектов, модель освещения и распространения света, которая отвечает требованиям конвейерной архитектуры графической системы.

В четвертой главе описывается вычисление векторов, необходимых для формирования теней и реалистичного закрашивания объектов сцены.

Каждая глава содержит упражнения для самостоятельной работы, призванные закрепить теоретический материал.

Следующий раздел пособия (раздел Б) является практико-ориентированным, и он посвящен основам программирования графических приложений в графической стандарте OpenGL.

Материал, изложенный в пособии, позволяет приступить к самостоятельной разработке несложных графических приложений. Для этого в пособии размещен лабораторный практикум.

Заключительная глава раздела Б содержит требования к оформлению отчета о проделанной работе.

Завершает пособие словарь специальных терминов и понятий.

В результате выполнения цикла лабораторных работ сформируются навыки разработки приложения, содержащего трехмерную анимационную сцену. Лабораторный практикум построен на принципе от простого к сложному. Добавление элементов и составляющих графический образ позволяет изучить основы компьютерной графики и одновременно продемонстрировать возможности графической библиотеки. Выбор графических объектов, характеристик источников света, свойств материалов и текстур осуществляется обучаемым самостоятельно, тем самым не ограничивает его творчество.

ВВЕДЕНИЕ

Стремительное развитие информационных и телекоммуникационных технологий не только изменило потребности человечества, но и повлияло на мировоззрение людей. В таких областях, как игровая индустрия, кинематография, средства массовой информации, финансовый сектор и образование, внедрение современных компьютерных технологий уже произвели поистине революционный переворот. Следующим шагом на этом пути становится интернет-вещей. Интеграция в единую систему компонентов компьютерной графики, сетевых технологий и систем машинного зрения открыла новые пути передачи и отображения информации. Массовая доступность электронных устройств и разработка приложений с использованием широчайших возможностей компьютерной графики и звука позволяет говорить о новом уровне организации взаимодействия человека и машины.

Компьютерная графика, как наука, начала развиваться около 50–60 лет назад. В те годы удавалось добиться отображения нескольких десятков отрезков на экране электронно-лучевой трубки, а современные графические системы позволяют синтезировать изображения, неотличимые по качеству от созданных профессиональной камерой. Общепринятой практикой стало обучение с помощью систем моделирования виртуальной реальности и симуляторов. Такие системы созданы для обучения пилотов самолета, водителей автотранспорта, машинистов поездов. Будущие врачи практикуются делать операции на специальном виртуальном оборудовании. На экран выходят полнометражные фильмы, в которых нет ни одного снятого кадра, все действие смоделировано в памяти компьютера. В области программного обеспечения также произошли серьезные изменения. Появились гра-

фические стандарты. Библиотека OpenGL (Open Graphic Library) стала стандартным интерфейсом для программистов как при написании прикладных программ, так и при разработке программных продуктов высокого класса — от интерактивных игр до систем визуализации результатов научных исследований.

Математическая база компьютерного моделирования существенно отличается от математической теории, используемой в инженерной графике. И на это есть свои причины.

Основное отличие связано с необходимостью различать математическое описание точки от вектора.

Из аналитической геометрии *точка* — абстрактный объект в пространстве, не имеющий никаких измеримых характеристик, кроме координат. *Вектор* (от лат. *vector* «несущий») — математический объект, характеризующийся величиной и направлением. В математической теории компьютерной графики понятие «вектор» трактуется как направленный отрезок прямой, представленный совокупностью *n* чисел. Все графические объекты компьютерной графики описываются множеством вершин. Под вершиной (англ. *vertex*) понимается структура данных, которая описывает положение точки в 2D- или 3D-пространстве.

Чтобы понять, как в программе формируются графические объекты, нужно знать методы представления в графической системе вершин (точек в пространстве).

Математическое обеспечение систем компьютерной графики — это совокупность методов представления и манипулирования множеством геометрических элементов, главными из которых являются точки и отрезки прямых [11]. Эти методы базируются на свойствах линейного (векторного), аффинного и евклидова пространства. Векторное пространство содержит объекты двух типов: скаляры и векторы. В аффинном пространстве к ним добавляется третий тип объектов — точка. В евклидовом пространстве вводится понятие расстояния.

Предполагаем, что читатели данного пособия имеют представление о скалярном поле и векторном пространстве. В векторном пространстве отсутствуют такие понятия, как положение и расстояние. Если в качестве пространства для решения геометрических задач компьютерной графики использовать векторное пространство, то мы встретимся со множеством проблем. Одна из них заключается в том, что векторы являются свободными, т. е. не имеют точки приложения. В результате в векторном пространстве отсутствует возможность опреде-

лить особую точку, к которой должны быть привязаны базисные векторы. Эта проблема решается вводом в векторное пространство еще одного типа объектов — точки. Такое «дополнение» векторного пространства получило название аффинного. В аффинном пространстве возникает понятие фрейма как совокупности точки отчета и базисных векторов. Отказ от знакомого со школьной скамьи понятия системы координат в пользу понятия фрейма позволяет избежать трудностей при работе с векторами. Более того, концепция фрейма позволяет использовать единый математический аппарат матричной алгебры для манипуляции с точками и векторами, сохраняя при этом различие между двумя типами геометрических объектов. Хотя аффинное пространство и содержит все элементы, необходимые для построения геометрических моделей, в нем отсутствует понятие расстояния между двумя точками, а следовательно, и понятие длины вектора. Аффинное пространство, дополненное понятием расстояния, превращается в евклидово.

В евклидовом пространстве вводится новая операция — скалярное произведение, операндами которой являются векторы, а результатом — скаляр. Операция скалярного произведения дает нам меру угла между двумя векторами.

Проблема различия представления вектора и точки привела к возникновению понятия однородных координат. В системе однородных координат для представления точки и векторов трехмерного пространства используются четырехмерные матрицы-столбцы. В пособии подробно описан математический аппарат и преимущества его использования для программирования компьютерной графики. Пользуясь математическим аппаратом однородных координат, можно выполнять операции над точками и векторами с помощью обычных операций матричной алгебры. Из пособия вы узнаете, почему при использовании четырех измерений количество арифметических операций даже снижается.

РАЗДЕЛ А. ОСНОВЫ ПОСТРОЕНИЯ КОМПЬЮТЕРНОЙ ГРАФИКИ

Глава 1. Математический аппарат

1.1. Фреймы

В трехмерном векторном пространстве любой вектор s можно однозначно представить в виде линейной комбинации трех линейно-независимых векторов v_1, v_2, v_3:

$$s = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 .$$

Скаляры α_1, α_2, α_3 — это координаты вектора s в базисе в v_1, v_2, v_3. Можно записать представление вектора в матричной форме

$$s = \boldsymbol{a}^T \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad \boldsymbol{a} = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix}.$$

При этом базис определяет конкретную систему координат. Мы привыкли рисовать систему координат на бумаге в виде трех векторов, которые исходят из одной точки (рис. 1.1, *а*). Однако векторы, как геометрические объекты, не имеют точки приложения, а характеризуются только направлением и модулем. Следовательно, с точки зрения математики векторы на рис. 1.1, *б* эквивалентны векторам рис. 1.1, *а*.

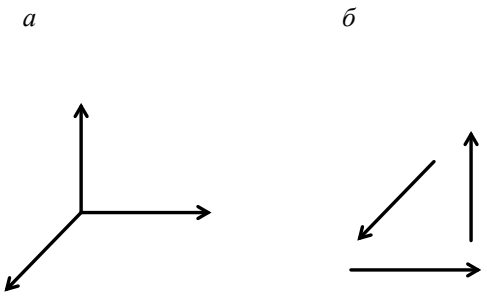

Рис. 1.1. Базисные векторы:
а — векторы, исходящие из одной точки; *б* — свободные векторы

Аффинное пространство включает понятие точки, следовательно, можно зафиксировать определенную точку отсчета — начало координат. Из этой точки можно изображать оси системы координат, что является общепринятым. Таким образом, мы переходим к понятию фрейма. *Фреймом* называется совокупность точки отсчета C_0 и базисных векторов, исходящих из этой точки.

В конкретном фрейме точно так же, как и в векторном пространстве, любой вектор можно однозначно записать в виде

$$s = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 .$$

Любую точку можно представить в виде

$$B = C_0 + \beta_1 v_1 + \beta_2 v_2 + \beta_3 v_3 .$$

Таким образом, во фрейме любой вектор описывается тремя скалярами, а любая точка представляется тремя скалярами и точкой отсчета. Понятие фрейма позволяет представить точки и векторы таким образом, что для манипулирования с ними можно использовать единый математический аппарат матричной алгебры, при этом сохранить различие между этими двумя типами объектов.

1.2. Однородная система координат

Представим точку B с координатами (x, y, z) с помощью трехмерного фрейма с началом в точке C_0 и базисом $\{v_1, v_2, v_3\}$

$$B = \begin{bmatrix} x \\ y \\ z \end{bmatrix},$$

где (x, y, z) — компоненты базисных векторов в этой точке.

Однако в этом случае точка будет представлена так же, как вектор, что для компьютерной графики неприемлемо. В математической литературе очень часто точка ассоциируется с вектором, проведенным в эту точку из начала координа В компьютерной графике такая ситуация может быть причиной различных недоразумений. Например, вектор из точки (0, 0, 0) в точку (2, 4, 6) равен вектору, проведенному из точки (2, 2, 2) в точку (4, 6, 8), поскольку оба вектора имеют одинаковый модуль и направление. Но второй вектор нельзя ассоциировать с точкой (2, 4, 6), пока его точка приложения не будет совмещена с началом координат — точкой (0, 0, 0).

Реализация графической системы требует однозначного представления точки и вектора, поскольку без этого невозможно представить смещение фрейма в пространстве и все виды трансформаций с помощью операции перемножения матриц.

Чтобы отличить точку от вектора, воспользуемся так называемыми однородными координатами[1]. В системе однородных координат для представления точек и векторов трехмерного пространства используются четырехмерные матрицы столбцы.

Пусть на множестве точек операция умножения на скаляры 0 и 1 определяется следующим образом:

$$0 \cdot \mathrm{C} = 0,$$
$$1 \cdot \mathrm{C} = \mathrm{C}.$$

Тогда точка *B* запишется в матричной форме с помощью перемножения матриц:

$$B = \begin{bmatrix} \beta_1 & \beta_2 & \beta_3 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ C_0 \end{bmatrix}.$$

Матрица-строка в правой части уравнения является представлением точки *B* в однородных координатах во фрейме с параметрами (v_1, v_2, v_3, C_0). Точку *B* можно представить матрицей-столбцом

$$B = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ 1 \end{bmatrix}.$$

[1] Однородные координаты были придуманы геометрами, и только значительно позже они стали применяться специалистами компьютерной графики.

В том же самом фрейме любой вектор можно представить в виде

$$s = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 = \begin{bmatrix} \alpha_1 & \alpha_2 & \alpha_3 & 0 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ C_0 \end{bmatrix}.$$

Значит, вектор во фрейме можно представить матрицей-столбцом

$$s = \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ 0 \end{bmatrix}.$$

Самое важное преимущество использования математического аппарата однородных координат состоит в том, что все аффинные преобразования, речь о которых пойдет ниже, выполняются единообразно, что позволяет довольно просто организовать выполнение сложных преобразований (анимацию, трансформацию объектов). Более того, несмотря на увеличение измерения (четыре измерения в трехмерном пространстве), количество операций даже уменьшается за счет возможности реализовать вычисления аппаратно и встроить их в графический конвейер.

1.3. Переход из одного фрейма в другой

При использовании трехмерного пространства возникает проблема изменения параметров фрейма. Плоскопараллельное смещение или перенос начала координат выполнить, изменяя базис в трехмерном пространстве, нельзя. Совсем другое дело в однородных координатах: за счет четвертой координаты становится возможным выполнять любое преобразование фрейма.

Известно, что переход из одного базиса в другой выполнятся с помощью системы линейных уравнений. Предположим, что имеется два базиса $\{v_1, v_2, v_3\}$ и $\{u_1, u_2, u_3\}$. Каждый вектор первого базиса можно представить во втором базисе и наоборот. Матричное уравнение имеет вид

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \mathrm{M} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}.$$

Матрица M содержит коэффициенты для преобразования координат вектора из одного базиса в другой. Рассмотрим преобразование из одного фрейма в другой в однородных координатах. Запишем систему линейных уравнений для преобразования из базиса $\{v_1, v_2, v_3\}$ в базис $\{u_1, u_2, u_3\}$

$$\begin{cases} u_1 = \nu_1, \\ u_2 = \nu_1 + \nu_2, \\ u_3 = \nu_1 + \nu_2 + \nu_3. \end{cases}$$

Если начало координат фрейма не меняет своего положения, то уравнения можно дополнить четвертым

$$P_0 = C_0.$$

Матрица преобразования имеет вид

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Если кроме изменения базиса изменяется и точка начала координат в новом фрейме, то четвертое уравнение будет содержать смещение начала координат, например:

$$P_0 = C_0 + \nu_1 + 2\nu_2 + 3\nu_3.$$

Матрица преобразования M примет вид

$$\mathrm{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 2 & 3 & 1 \end{bmatrix}.$$

Вычислить транспонированную, а затем обратную матрицу для обратного преобразования не составит труда.

$$\mathrm{K} = \left(\mathrm{M}^T\right)^{-1} = \begin{bmatrix} 1 & -1 & 0 & 1 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 1 & -3 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Таким образом, мы имеем матрицы преобразования разных фреймов в обе стороны.

Продемонстрируем на примере отличия в результатах преобразование точки и вектора. Уравнение преобразования имеет вид

$$b = Ka.$$

Представление точки в однородных координатах

$$a = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 1 \end{bmatrix}.$$

После преобразования получим

$$a' = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Получили начало координат нового фрейма. Матрица K преобразует точку b с координатами (1, 2, 3) в точку (0, 0, 0).

Вектор с такими же координатами в однородных координатах исходного фрейма имеет вид

$$x = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 0 \end{bmatrix};$$

преобразуется в новом фрейме в вектор

$$y = \begin{bmatrix} -1 \\ -1 \\ 3 \\ 0 \end{bmatrix}.$$

При работе с графикой обычно используют два фрейма: фрейм камеры и мировой фрейм. Фиксированный может быть любой из фреймов. Если зафиксировать фрейм камеры (камера взгляда неподвижна, а объекты изменяют свое положение), то матрица вида задает поло-

жение мирового фрейма и преобразует представления точек и векторов в однородных координатах в мировом фрейме в представление во фрейме камеры. Другие фреймы, необходимые для размещения объектов сцены, формируются с помощью однородных преобразований относительно фрейма камеры. Графическая система позволяет сохранять текущее значение матрицы. Таким образом, у программиста имеется возможность переключаться между фреймами, просто изменяя текущее значение матрицы вида. Как задать матрицу вида в графической библиотеке OpenGL, описано в разделе Б.

1.4. Аффинные преобразования

Под преобразованием будем понимать функцию, которая отображает точку (вектор) в другую точку (вектор). Графический смысл преобразования представлен на рис. 1.2.

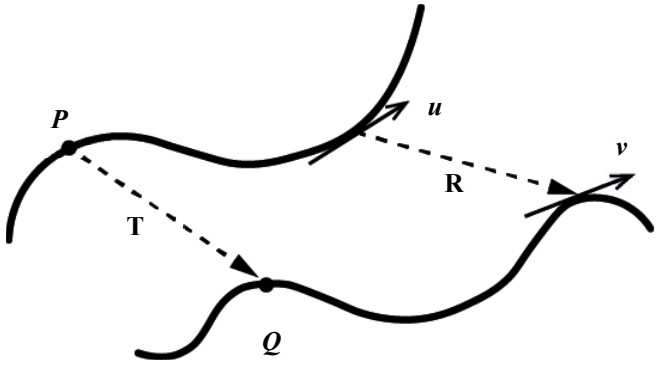

Рис. 1.2. Преобразование кривой

В компьютерной графике преобразование рассматривается как способ перемещения группы вершин, описывающих один или несколько объектов, в новую позицию.

В четырехмерном пространстве однородных координат векторы и точки представляются в виде матриц-столбцов. Преобразование для указанных объектов одного и того же фрейма имеет единообразный вид

$$p = f(q),$$

$$v = f(u).$$

Будем рассматривать ограниченный класс преобразований. В этом классе наложены ограничения на вид функции $f()$. Самое важное огра-

ничение — линейность функции. Функция $f()$ является линейной тогда и только тогда, когда для любых скаляров α и β и для любых точек p и q выполняется соотношение

$$f(\alpha p + \beta q) = \alpha f(p) + \beta f(q).$$

Преимущество такого преобразования в том, что если известно преобразование вершин, то всегда можно получить и преобразование линейной комбинации вершин.

Линейное преобразование всегда может быть записано в виде перемножения матриц

$$v = Au,$$

где A — квадратная матрица. Такое преобразование приведет к преобразованию фрейма. Следовательно, линейное преобразование можно рассматривать двояко: с одной стороны, как изменение фрейма, и с другой — как преобразование вершин в одном и том же фрейме. В однородных координата матрица A имеет вид

$$A = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Двенадцать элементов матрицы могут изменяться произвольно, о такой матрице говорят, что она имеет 12 степеней свободы. Однако если рассмотреть применение произвольного преобразования по отношению к вектору в однородных координатах, то на результат будут влиять только 9 элементов матрицы A, т. е. преобразование векторов обладает только девятью степенями свободы.

Покажем, что аффинное преобразование преобразует прямую линию в прямую линию. Пусть прямая записана в виде

$$P(\alpha) = P_0 + \alpha d,$$

где P_0 — точка, а d — вектор. В некотором фрейме точка P_0 описывается с помощью представления $\boldsymbol{p}_0$, а вектор d — с помощью представления $\mathbf{d}$ соответственно.

$$\mathbf{p}(\alpha) = \mathbf{p}_0 + \alpha \mathbf{d}.$$

Для любой матрицы аффинного преобразования $\mathbf{A}$ справедливо соотношение

$$\mathbf{Ap}(\alpha) = \mathbf{Ap}_0 + \alpha\mathbf{Ad}.$$

Таким образом, можно построить преобразование прямой, выполнив сначала преобразование точки, затем вектора, а далее воспользоваться алгоритмом формирования прямой. Рассуждения в терминах абстрактного математического пространства имеют конкретный практический смысл, а именно: для построения преобразованного отрезка необходимо выполнить только преобразование в однородных координатах конечных точек такого отрезка. Из этого следует, что графическую систему можно реализовать в виде конвейера аффинных преобразований представлений отдельных точек в однородных координатах.

К аффинным преобразованиям относятся следующие преобразования:

1) плоскопараллельное смещение, или сдвиг (*translation*), — преобразование смещает точки на фиксированное расстояние вдоль заданного направления (рис. 1.3). Сдвиг задается вектором смещения и имеет три степени свободы;

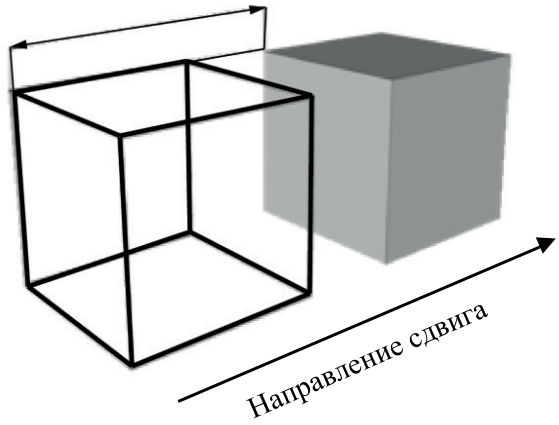

Рис. 1.3. Преобразование «Сдвиг»

2) поворот (*rotation*) — преобразование, при котором точки перемещаются на некоторый угол в заданной плоскости вращения относительно фиксированного центра или оси вращения. Для преобразования поворота необходимо определить три параметра: фиксированную точку P, угол поворота θ, вектор u, вокруг которого выполняется поворот (ось вращения), (рис. 1.4);

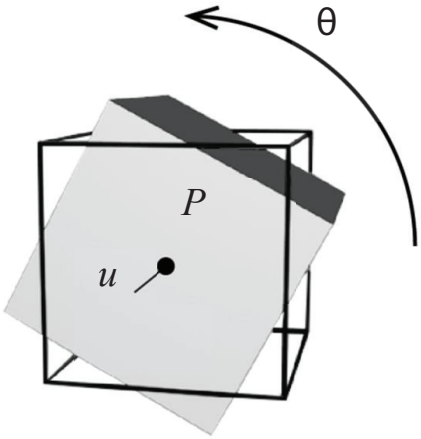

Рис. 1.4. Преобразование «Поворот»

3) масштабирование (*scaling*) — преобразование увеличивает или уменьшает размеры объекта. Масштабирование задается направлением, вдоль которого изменяется масштаб и масштабный множитель α (коэффициент гомотетии). При $\alpha > 1$ объект растягивается в заданном направлении, а при $0 \le \alpha < 1$ объект сжимается. При отрицательном значении α происходит отражение объекта относительно заданной фиксированной точки в указанном направлении. На рис. 1.5 показано отражение объекта в направлении вертикальной оси, $\alpha = -1$.

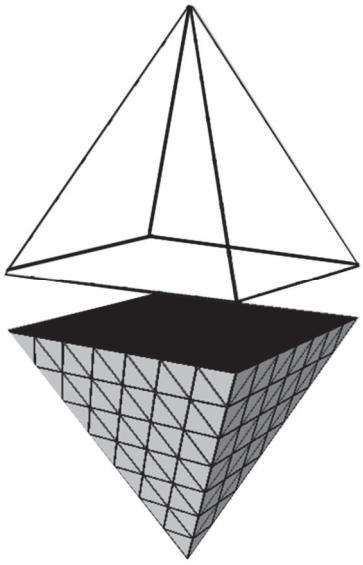

Рис. 1.5. Преобразование «Отражение»

1.5. Преобразования в однородных координатах

В некотором фрейме любое аффинное преобразование может быть представлено матрицей 4×4 вида

$$A = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Преобразование сдвига смещает точку в новые позиции в соответствии с заданным вектором смещения. Пусть точка $P_1(x_1, y_1, z_1)$ смещается на расстояние $\boldsymbol{d}(\alpha, \beta, \gamma)$, тогда координаты преобразованной точки можно описать системой уравнений

$$\begin{cases} x_2 = x_1 + \alpha, \\ y_2 = y_1 + \beta, \\ z_2 = z_1 + \gamma. \end{cases}$$

Однако представление преобразования сдвига в виде суммирования матриц-столбцов не позволяет унифицировать любое аффинное преобразование.

Другой метод представления сдвига — с помощью перемножения матриц. Этот метод возможен только в однородных координатах. К системе уравнений добавляем четвертое уравнение

$$\begin{cases} x_2 = x_1 + \alpha, \\ y_2 = y_1 + \beta, \\ z_2 = z_1 + \gamma, \\ 1 = 1. \end{cases}$$

Полученную систему можно представить в матричном виде

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \alpha \\ 0 & 1 & 0 & \beta \\ 0 & 0 & 1 & \gamma \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}.$$

Матрица $\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & \alpha \\ 0 & 1 & 0 & \beta \\ 0 & 0 & 1 & \gamma \\ 0 & 0 & 0 & 1 \end{bmatrix}$ называется *матрицей сдвига (translation matrix)*. Применение однородных координат для выполнения преобразования сдвига рассматривается как математический трюк, позволяющий заменить суммирование трехмерных матриц-столбцов перемножением четырехмерных матриц.

Матрицу обратного преобразования можно интерпретировать как сдвиг на расстояние $-\boldsymbol{d}(-\alpha, -\beta, -\gamma)$.

$$\mathbf{T}^{-1} = \begin{bmatrix} 1 & 0 & 0 & -\alpha \\ 0 & 1 & 0 & -\beta \\ 0 & 0 & 1 & -\gamma \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Преобразования поворота и масштабирования характеризуются фиксированной точкой, которая при выполнении преобразований остается неподвижна. Будем считать, что фиксированной точкой является начало координат.

Система уравнений, задающая преобразование масштабирования, имеет вид

$$\begin{cases} x_2 = \alpha x_1, \\ y_2 = \beta y_1, \\ z_2 = \gamma z_1, \\ 1 = 1, \end{cases}$$

где α, β, γ — масштабные коэффициенты по каждой из координатных осей.

В матричной форме

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}.$$

Матрица $\mathbf{S} = \begin{bmatrix} \alpha & 0 & 0 & 0 \\ 0 & \beta & 0 & 0 \\ 0 & 0 & \gamma & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$ называется матрицей масштабирования (*scale matrix*).

Для обращения матрицы масштабирования необходимо использовать обратные значения масштабных коэффициентов по осям:

$$\mathbf{S}^{-1}(\alpha, \beta, \gamma) = S\left(\frac{1}{\alpha}, \frac{1}{\beta}, \frac{1}{\gamma}\right).$$

Преобразование поворота имеет три степени свободы, так как его можно выполнять независимо вокруг каждой из осей координат. Запишем в однородных координатах систему уравнений, представляющую поворот точки $P_1(x_1, y_1, z_1)$ вокруг оси Oz на угол θ. При этом компонент z точки не изменится

$$\begin{cases} x_2 = x_1\cos\theta + y_1\sin\theta, \\ y_2 = x_1\sin\theta - y_1\cos\theta, \\ z_2 = z_1, \\ 1 = 1. \end{cases}$$

В матричном виде имеем

$$\begin{bmatrix} x_2 \\ y_2 \\ z_2 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix}.$$

Аналогично этой матрице можно сформировать матрицы поворота вокруг осей Ox и Oy. Матрицы поворотов имеют вид

$$\mathbf{R}_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\mathbf{R}_y = \begin{bmatrix} \cos\theta & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Вернуть точку в исходное состояние можно, выполнив поворот вокруг той же оси на угол $(-\theta)$. Следовательно, $\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta)$. При рассмотрении матриц видно, что элементы, содержащие косинус, всегда размещаются на главной диагонали матрицы, а пара элементов, содержащих синус, — по разные стороны от главной диагонали.

Из тригонометрии известно $\cos(-\theta) = \cos\theta$ и $\sin(-\theta) = -\sin\theta$, поэтому получим

$$\mathbf{R}^{-1}(\theta) = \mathbf{R}^{\mathrm{T}}(\theta).$$

Матрицы, для которых операция транспонирования дает тот же результат, что и обращение, называются *ортогональными*. Любая ортогональная матрица описывает некоторое преобразование поворота с точкой в начале координат.

Матрица сложного аффинного преобразования формируется перемножением матриц базовых преобразований, т. е. сложное преобразование есть суперпозиция сдвига, поворота и масштабирования. Такой подход очень хорошо сочетается с конвейерной архитектурой графической системы. Пусть выполняется три последовательных преобразования (**A**, **B**, **C**) точки ***p***, при этом формируется новая точка ***q*** (рис. 1.6).

Рис. 1.6. Сложное преобразование

Учитывая, что произведение матриц ассоциативно, запишем

$$\boldsymbol{q} = \mathrm{CBA}\boldsymbol{p}.$$

Порядок выполнения операций существенно влияет на производительность процесса вычисления. Можно выполнять преобразования последовательно, при этом на каждом шаге квадратная матрица умножается на матрицу-столбец, а можно сначала вычислить матри-

цу единого преобразования, а затем умножать ее на матрицу-столбец.

В случае, когда нужно обработать множество точек, второй подход является более экономичным с точки зрения вычислительной сложности.

В графическом конвейере сначала вычисляется матрица произведения **M** = CBA, а затем она загружается в ту часть конвейера, которая выполняет преобразования для множества точек (рис. 1.7).

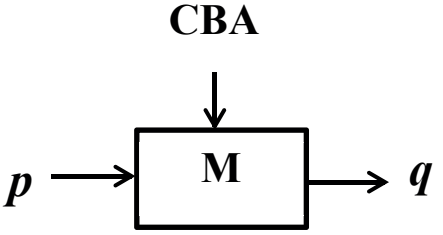

Рис. 1.7. Выполнение преобразований в графическом конвейере

1.5. Преобразование фрейма в графической системе

Для преобразования фрейма в графических системах используется текущая матрица преобразования (CTM — *current transformation matrix*), или матрица вида. Эта матрица 4×4 применяется для преобразования всех вершин фрейма. Изменение матрицы влечет за собой некоторое аффинное преобразование. Матрицу вида можно изменить двумя способами: присвоением новых значений элементам матрицы и умножением ее на другую матрицу. В исходном состоянии матрица CTM является единичной, при необходимости в любой момент времени прикладная программа может ее инициализировать.

Упражнения к главе 1

1. Докажите, что следующие преобразования коммутативны:
   - поворот и равномерное масштабирование;
   - два поворота вокруг одной и той же оси;
   - два смещения.
2. Сколько точек нужно использовать для нахождения матрицы аффинного преобразования в трехмерном пространстве?
3. Найдите матрицу поворота вокруг произвольной фиксированной точки.
4. Выведите матрицу преобразования скоса.
5. Найдите аффинное преобразование, которое формирует отражение точки относительно прямой $y = ax + b$.

6. Матрица поворота вокруг произвольной оси может быть получена в результате суперпозиции трех матриц поворотов вокруг координатных осей. Сколько способов подобного разложения существует? Всегда ли нужно использовать все три оси координат?

7. Как представить плоскость в однородных координатах?

8. Опишите матричное представление геометрических преобразований в однородных координатах.

Глава 2. Проецирование

2.1. Классическая и компьютерная визуализация

В этой главе будет рассмотрен интерфейс между системой компьютерной графики и прикладной программой, необходимый для визуализации трехмерных объектов. При использовании модели синтезированной камеры (искусственно созданной компьютером) необходимо отметить сходство классических и компьютерных методов визуализации. Базовые элементы в обоих случаях одни и те же: объекты, проецирующие лучи и картинная плоскость (плоскость проекции) (рис. 2.1).

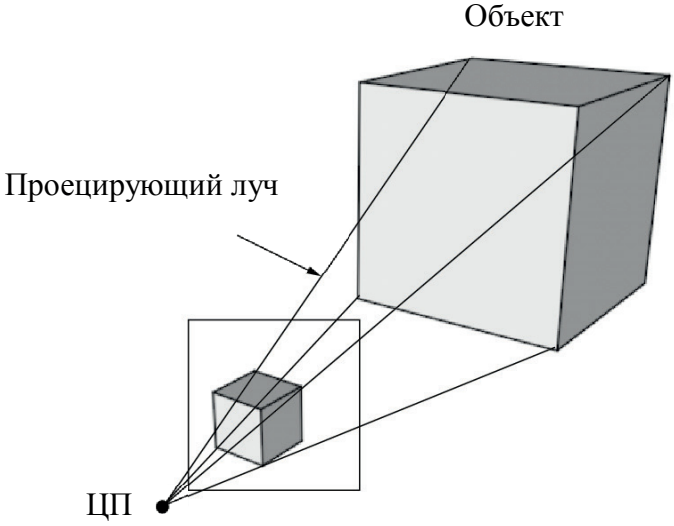

Рис. 2.1. Визуализация

Проецирующие лучи пересекаются в точке, которая называется центром проецирования (ЦП). Точка ЦП соответствует фокусу объектива или глаза и в системе компьютерной графики выбирается в качестве центра фрейма камеры (camera frame). Модель синтезированной камеры, базирующаяся на геометрической оптике, положена в основу всех стандартных графических систем. Однако следует понимать, что использование плоскости в качестве поверхности, на которую проецируется изображение, не является единственно возможным вариантом. Иногда изображение проецируется на цилиндрическую или сферическую поверхность, например как отражение на некотором объекте, но в стандартных графических системах такие варианты не учитываются.

Как в классической, так и в компьютерной графике ЦП может быть расположен достаточно далеко от картинной плоскости и от объектов, что делает проецирующие лучи параллельными. В этом случае центр проецирования в модели заменяется направлением проецирования (*DOP — direction of projection*), как показано на рис. 2.2.

Виды, которые создаются при расходящихся проецирующих лучах, т. е. когда ЦП находится не слишком далеко от картинной плоскости, называются перспективными (*perspective views*), а при параллельных проецирующих лучах — параллельными видами (*parallel views*).

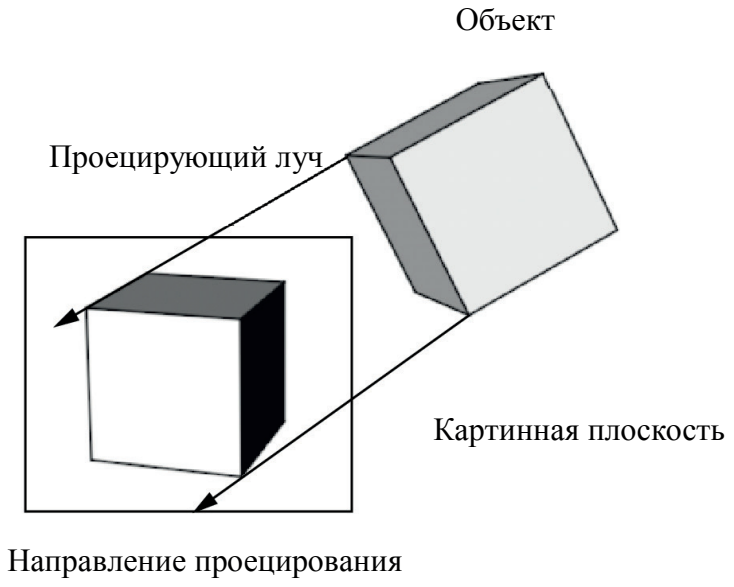

Рис. 2.2. Удаление ЦП в бесконечность

В классической графике, кроме параллельной и перспективной проекций, используется и множество других способов проецирования, начиная с классической системы трех ортогональных видов и заканчивая двух- и трехточечными перспективными видами. Такие виды позволяют передать зависимость между объектом, наблюдателем и картинной плоскостью. В компьютерной графике принято объект и наблюдателя воспринимать независимо друг от друга, поэтому множество видов проецирования не используется в процессе визуализации.

Те, кто увлекаются фотографией, знают, что перед фотографированием необходимо настроить фокусное расстояние трансфокатора[2]. Размеры кадра зависят от размера матрицы и характеристик объектива. В компьютерной графике имеет место аналогичная ситуация: какие объекты будут включены в изображение (попадут в кадр), зависит от выбора типа проецирования и параметров визуализации.

В фотографии широкоугольный объектив дает наиболее ощутимую перспективу: объекты, расположенные близко к камере, имеют на снимке большие размеры, чем объекты, несколько удаленные от камеры. И наоборот, длиннофокусные объективы (их часто называют телеобъективами) скрывают перспективу, уравнивая размеры объектов. Говорят, что такое изображение «плоское», приближающееся по характеру к тому, что создается при параллельном проецировании.

В большинстве графических API параллельное и перспективное проецирование рассматриваются как два разных типа, причем каждому из них соответствует своя функция обработки и набор парметров.

Поскольку моделирование проецирования является ключевой задачей в трехмерной компьютерной графике, без глубокого понимания используемых при этом математических методов нельзя ни написать хорошую прикладную программу, ни разработать новую графическую систему. Более того, хотя функции работы с проективными преобразованиями, которые имеются в графических библиотеках, и дают вполне удовлетворительные результаты в большинстве ситуаций, возникающих в практике разработки графических приложений, некоторые виды проекций, в частности косоугольные, чаще всего не поддерживаются. Если в приложении нужно организовать именно такую проекцию, приходится прибегать к косвенным методам, а для этого необходимо глубокое понимание механизмов работы системы.

[2] Объектив с переменным фокусным расстоянием.

2.2. Проективное преобразование

Процесс проецирования можно рассмотреть как преобразование точки (x, y, z) в другую точку (x_p, y_p, z_p), лежащую на картинной плоскости. *Проективное преобразование* — отображения точки трехмерного пространства в точки картинной плоскости (рис. 2.3). Такое преобразование является вырожденным, т. е. определитель матрицы преобразования равен нулю. Все точки, расположенные на одном проецирующем луче, преобразуются в единственную точку на картинной плоскости. По этой причине восстановить исходной положение нельзя, следовательно, преобразование является необратимым.

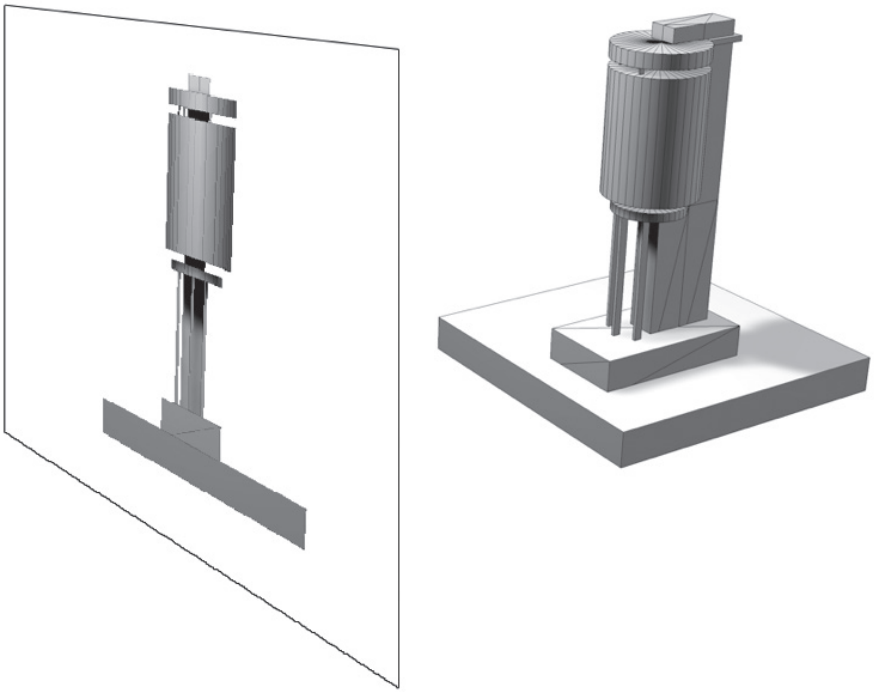

Рис. 2.3. Проективное преобразование

Кроме того, необходимо учитывать: фокусное расстояние (расстояние от ЦП до картинной плоскости), размеры окна картинной плоскости и угол обзора. Если картинная плоскость имеет вид прямоугольника, то на изображении появятся только те объекты, которые окажутся внутри пирамиды с вершиной в ЦП. Эта пирамида называется *зоной видимости*. Объекты, не попавшие в зону видимости, отсекаются и не включаются в кадр.

В системах компьютерной графики зона видимости ограничивается не только с боков, но и вдоль оси проецирования: задается ближняя и дальняя плоскости отсечения. В результате зона приобретает вид *усеченной пирамиды видимости* (рис. 2.4). В начале координат фрейма находится центр проецирования, единственный жестко зафиксированный параметр.

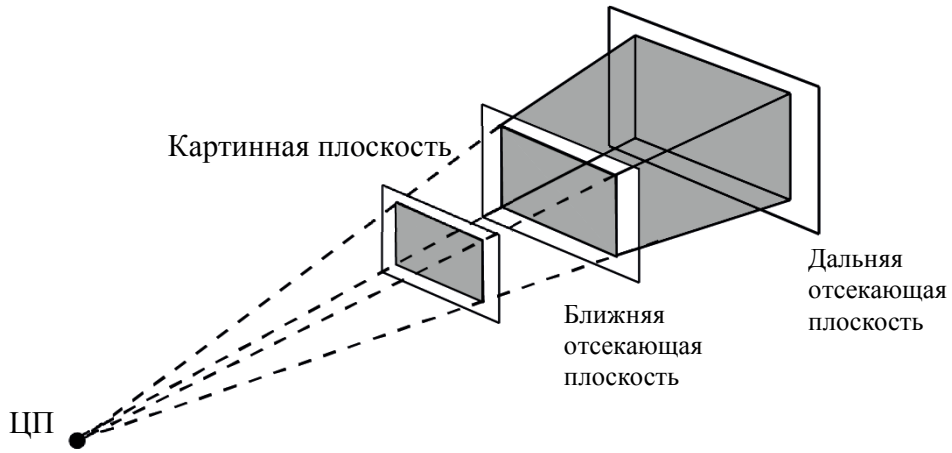

Рис. 2.4. Пирамида видимости

В большинстве графических API параметры отсечения определяются вместе со способом проецирования. Совсем не обязательно, чтобы параметры, задающие левую, правую, верхнюю и нижнюю грани пирамиды видимости, были симметричны относительно оси *z*.

При параллельном проецировании пирамида видимости превращается в *параллелепипед видимости*.

Каков бы ни был способ проецирования, результат необходимо нормализовать, т. е. привести к единому виду.

Подход, используемый в графической системе, базируется на методике *нормализации проецирования*. Данная методика предусматривает сведение всех типов проекций к ортогональной проекции. Для этого выполняется предварительное искажение исходных объектов (рис. 2.5). Поскольку такое искажение может быть представлено в виде преобразования в однородных координатах, можно не выполнять фактическое искажение формы объектов, а добавить соответствующие матрицы преобразований к матрице обычного ортогонального проецирования и получить матрицу требуемого проективного преобразования, как показано на рис. 2.6.

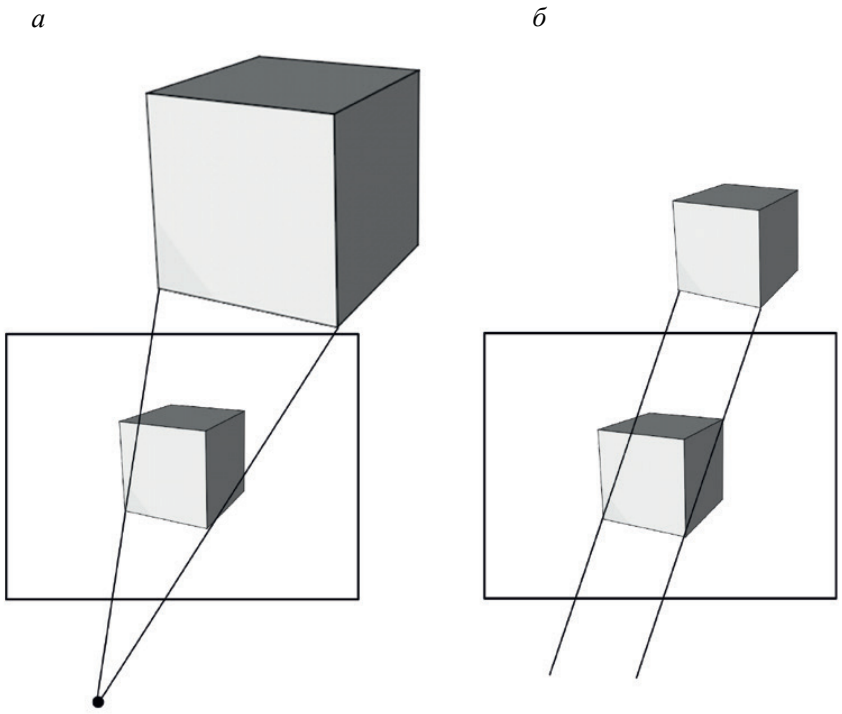

Рис. 2.5. Предварительное искажение объекта:
а — перспективная проекция; *б* — ортогональная проекция искаженного объекта

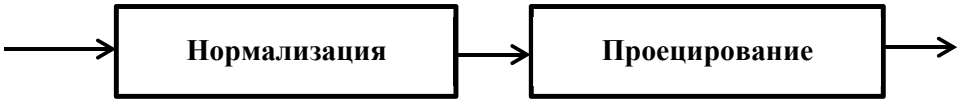

Рис. 2.6. Этапы проективного преобразования

2.3. Матрица параллельного проецирования

Несмотря на то что параллельная проекция является частным случаем перспективной проекции, сначала рассмотрим методику нормализации с параллельной ортогональной проекцией[3], а затем распространим ее на перспективную проекцию.

Процесс проецирования разделим на две части: сначала с помощью невырожденного преобразования в однородных координатах преобразуем заданную зону видимости в стандартную (нормализация), а за-

[3] Ортогональная проекция — это частный случай параллельной проекции, при которой проецирующие лучи ортогональны картиной плоскости.

тем выполним проективное преобразование. На первом этапе объекты искажаются таким образом, что после выполнения на втором этапе проективного преобразования выбранного типа (ортогональное) в отношении этих объектов и зоны видимости получим

$$x_p = x, y_p = y, z_p = 0 .$$

По сути ортогональное проективное преобразование сводится к приравниванию компонента z к нулю или к его игнорированию, поскольку при отображении он становится ненужным. Таким образом, вся вычислительная нагрузка переносится на первый этап процесса.

Разделение процесса проецирования на два этапа в значительной мере связано с другими задачами компьютерной графики, выполняемыми в ходе конвейерной обработки. В частности, операция отсечения[4] выполняется при визуализации трехмерных объектов на нескольких этапах, а использование невырожденных матриц преобразования позволяет сохранять информацию о глубине расположения объектов вдоль проецирующего луча, которая в дальнейшем необходима для удаления невидимых поверхностей и правильного закрашивания объектов.

В OpenGL существует разделение между координатами экрана (*screen coordinates*), которые являются двухмерными и не содержат информации о глубине, и координатами окна (*window coordinates*), которые являются трехмерными и сохраняют информацию о глубине. Матрица проецирования и последующее перспективное деление преобразуют вершины в координаты окна.

Матрица проецирования (*Projection matrix*) формируется как суперпозиция двух аффинных преобразований: сдвига и масштабирования. При ортогональном проецировании простейшей зоной видимости является куб, центр которого находится в начале координат, а грани образуются плоскостями

$$x = 1, x = -1, y = 1, y = -1, z = 1, z = -1.$$

В OpenGL такой куб видимости устанавливается системой по умолчанию. Для его принудительной установки нужно включить в программу такую последовательность команд:

```
glMatrixMode (GL_PROJECTION);
glLoadIdentity ();
glOrtho (-1.0, 1.0,—1.0, 1.0,—1.0, 1.0);
```

[4] Отсечение — удаление ненужной графической информации.

Этот куб называется *канонической зоной видимости*. Два последних аргумента в вызове `glOrtho ()` — это расстояние до ближней и дальней отсекающих плоскостей, измеренное от позиции камеры в направлении отрицательной полуоси z. Передняя отсекающая плоскость $z = 1.0$ находится за камерой, задняя отсекающая плоскость $z = -1.0$ — перед камерой, хотя проецирующие лучи параллельны и ортогональная проекция по своим свойствам аналогична использованию телеобъектива с очень большим фокусным расстоянием.

Предположим, что при вызове функции `glOrtho ()` заданы произвольные значения аргументов:

`glOrtho` $(x_{\min}, x_{\max}, y_{\max}, y_{\min}, l_{\text{near}}, l_{\text{far}})$;

Такой набор аргументов задает правильный параллелепипед видимости, правая сторона которого, если смотреть от камеры, — это плоскость $x = x_{\max}$, левая сторона — плоскость $x = x_{\min}$, верхняя грань — плоскость $y = y_{\max}$, а нижняя грань — плоскость $y = y_{\min}$. Ближняя и дальняя отсекающие плоскости — это $z = -l_{near}$ и $z = -l_{far}$. Матрица проецирования, которая формируется при этом, преобразует этот параллелепипед в куб, центр которого расположен в начале координат, а ребра имеют длину 2 единицы (рис. 2.7).

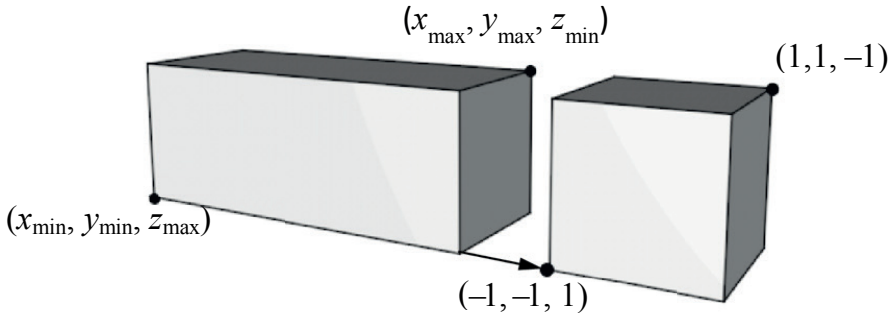

Рис. 2.7. Схема преобразования заданного параллелепипеда видимости в каноническую зону видимости

На рис. 2.7 $z_{\max}$ — взятое с обратным знаком значение аргумента l_{far}, а $z_{\min}$ — взятое с обратным знаком значение аргумента l_{near} вызова функции `glOrtho ()`.

Полученная таким способом матрица преобразует координаты вершин, описывающих объекты, которые были заданы при вызове `glVertex ()`, в вершины внутри канонической зоны видимости, для чего выполняются преобразования масштабирования и сдвига. В ре-

зультате этого те вершины, которые находились внутри заданного параллелепипеда видимости, преобразуются в вершины, находящиеся внутри канонической зоны видимости, а те вершины, которые были снаружи от заданного параллелепипеда видимости, окажутся и снаружи канонической зоны видимости.

Итак, матрица проецирования определяется заданным типом проекции и параметрами зоны видимости, переданными в качестве аргументов функции `glOrtho ()`. Положение и ориентация камеры задаются матрицей вида. Эти две матрицы — вида и проецирования — перемножаются, и координаты точек объектов умножаются справа на результирующую матрицу.

Для вывода матрицы ортогонального проецирования необходимо решить две задачи: сначала с помощью преобразования сдвига передвинуть центр заданного параллелепипеда в центр канонической зоны видимости — начало координат, затем изменить масштабные коэффициенты таким образом, чтобы каждое ребро параллелепипеда имело длину 2 единицы (рис. 2.8).

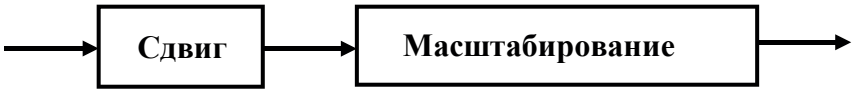

Рис. 2.8. Аффинные преобразования для нормализации

Имеем два преобразования

$$\mathbf{T}\left(-\frac{x_{\max}+x_{\min}}{2},-\frac{y_{\max}+y_{\min}}{2},-\frac{z_{\max}+z_{\min}}{2}\right)$$

$$\mathbf{S}\left(\frac{2}{x_{\max}-x_{\min}},\frac{2}{y_{\max}-y_{\min}},\frac{2}{z_{\max}-z_{\min}}\right),$$

после суперпозиции которых сформируется матрица проецирования

$$\mathbf{P}=\mathbf{ST}=\begin{bmatrix}\frac{2}{x_{\max}-x_{\min}} & 0 & 0 & -\frac{x_{\max}+x_{\min}}{x_{\max}-x_{\min}}\\ 0 & \frac{2}{y_{\max}-y_{\min}} & 0 & -\frac{y_{\max}+y_{\min}}{y_{\max}-y_{\min}}\\ 0 & 0 & \frac{2}{z_{\max}-z_{\min}} & -\frac{z_{\max}+z_{\min}}{z_{\max}-z_{\min}}\\ 0 & 0 & 0 & 1\end{bmatrix}.$$

Поскольку ось проецирования камеры направлена вдоль отрицательной полуоси z и проецирующие лучи направлены из бесконечности отрицательного полупространства в начало координат, то $z_{\max} = l_{far}$, а $z_{\min} = l_{near}$.

После вызова функции `gIOrtho ()` матрица проецирования примет следующий вид:

$$P = \begin{bmatrix} \frac{2}{x_{\max} - x_{\min}} & 0 & 0 & -\frac{x_{\max} + x_{\min}}{x_{\max} - x_{\min}} \\ 0 & \frac{2}{y_{\max} - y_{\min}} & 0 & -\frac{y_{\max} + y_{\min}}{y_{\max} - y_{\min}} \\ 0 & 0 & \frac{2}{l_{far} - l_{near}} & -\frac{l_{far} + l_{near}}{l_{far} - l_{near}} \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

2.4. Матрица перспективного проецирования

Для проективного преобразования, использующего перспективное проецирование, сначала нужно решить, какую форму должна иметь каноническая зона видимости. Затем следует сформировать преобразование перспективной нормализации (*perspective-normalization transformation*), которое преобразует перспективную проекцию в ортогональную. И, наконец, нужно вывести матрицу перспективного проецирования.

Для формирования изображения нам нужно знать параметры зоны видимости. Предположим, что выбран угол обзора 90°, $x_{\max} = 1$, $x_{\min} = -1$, $x_{\max} = 1$, $y_{\min} = -1$. Таким образом, боковые грани зоны видимости наклонены по отношению к картинной плоскости на 45°. Следовательно, зона видимости имеет вид пирамиды, бесконечной в одном направлении (рис. 2.9). Данную зону можно ограничить, задав ближнюю и дальнюю отсекающие плоскости $z = z_{\min}$ и $z = z_{\max}$, где оба значения отрицательны, причем $z_{\max} < z_{\min}$.

Рассмотрим матрицу

$$\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{bmatrix}.$$

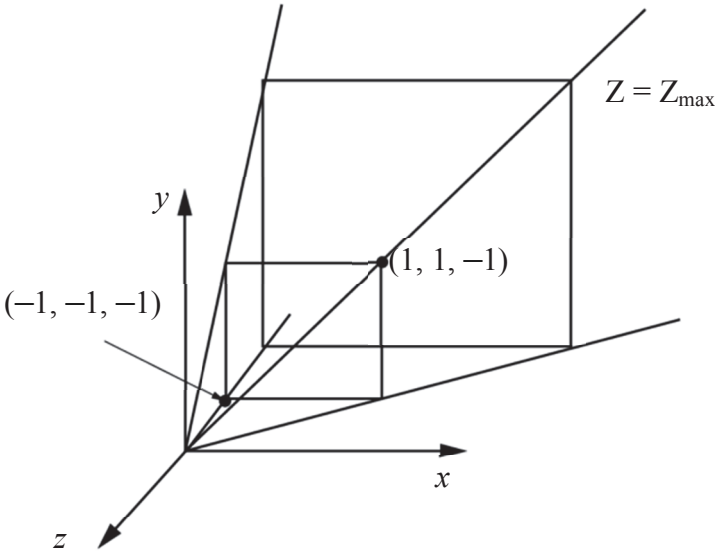

Рис. 2.9. Простая перспективная проекция

Матрица **N** не является вырожденной, если значения α и β отличны от 0. Если применить преобразование **N** к точке в однородных координатах $\mathrm{p} = [x\, y\, z\, 1]^{\mathrm{T}}$, то получим новую точку

$$\mathbf{q} = [x'\ y'\ z'\ w']^{\mathrm{T}},$$

где $x' = x, y' = y, z' = \alpha z + \beta, w' = -z.$

После деления всех компонентов вектора на w' получим координаты точки в трехмерном пространстве

$$x'' = -\frac{x}{z}, y'' = -\frac{y}{z}, z'' = -\left(\alpha + \frac{\beta}{z}\right).$$

Если применить по отношению к **N** преобразование ортогонального проецирования вдоль оси z, получим в результате следующую матрицу

$$\mathbf{M}_{ort}\mathbf{N} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix},$$

которая является матрицей простого перспективного преобразования.

Проекция произвольной точки **p** имеет вид

$$\mathbf{p}' = \mathbf{M}_{ort}\mathbf{N}\mathbf{p} = \begin{bmatrix} x \\ y \\ 0 \\ -z \end{bmatrix}.$$

Выполнив перспективное деление, получим координаты точки, лежащей на картинной плоскости:

$$x_p = -\frac{x}{z}, y_p = -\frac{y}{z}.$$

Следовательно, если применить к произвольной точке преобразование **N**, то ее ортогональная проекция будет иметь такой же вид, что и перспективная. Матрица **N** преобразует исходную зону видимости в новую. От параметров α и β зависит вид этой зоны. Необходимо выбрать параметры так, чтобы она была канонической.

Ближняя отсекающая плоскость зоны видимости $z = z_{\min}$ преобразуется в плоскость

$$z'' = -\left(\alpha + \frac{\beta}{z_{\min}}\right).$$

Дальняя отсекающая плоскость зоны видимости $z = z_{\max}$ преобразуется в плоскость

$$z'' = -\left(\alpha + \frac{\beta}{z_{\max}}\right).$$

Если выбрать значения параметров α и β следующими:

$$\alpha = -\frac{z_{\max} + z_{\min}}{z_{\max} - z_{\min}}, \beta = -\frac{2z_{\max}z_{\min}}{z_{\max} - z_{\min}},$$

то отображением плоскости $z = z_{\min}$ будет плоскость $z'' = 0$, плоскости $z = z_{\max}$ — плоскость $z'' = -1$, т. е. мы получим искомую каноническую зону видимости. На рис. 2.10 показано, как выглядит это преобразование и искажение куба внутри зоны видимости.

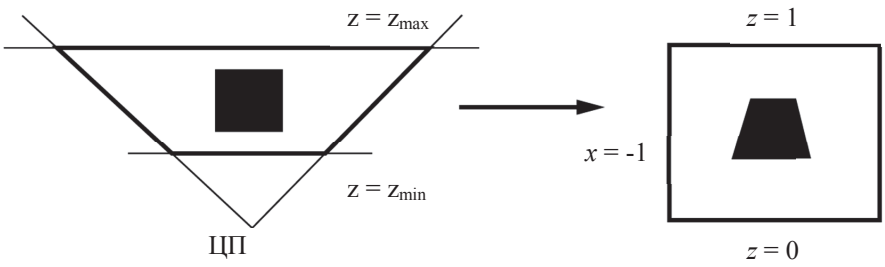

Рис. 2.10. Перспективная нормализация зоны видимости

Таким образом, матрица **N** трансформирует усеченную пирамиду видимости в правильный параллелепипед, а ортогональное преобразование трансформирует параллелепипед в канонический куб видимости. В результате ортогональная проекция искаженного объекта будет иметь тот же вид, что и перспективная проекция исходного. Матрицу **N** называют матрицей перспективной нормализации (*perspective-normalization matrix*).

Итак, мы показали, как с помощью предварительного искажения можно свести и перспективное, и параллельное проективное преобразование к ортогональному. Такая замена позволяет включить любой тип проективного преобразования в стандартный процесс конвейерной обработки.

2.5. Проецирование и формирование теней

Одна из задач, при решении которой используются матрицы проективного преобразования, — формирование теней в изображении. Несмотря на то что в структурном смысле тени и не являются геометрическими объектами, без них невозможно представить себе, как передать в плоском изображении трехмерную форму объектов. Тени возникают только в том случае, если имеется хотя бы один источник света. Некоторая точка на объекте оказывается в тени, если на нее не попадает свет ни от одного из имеющихся источников света. Но если источник находится в центре проецирования, то мы не увидим теней, поскольку все точки, находящиеся в тени, закрыты видимыми поверхностями объектов. Такая модель освещения называется моделью «засветки».

Для того чтобы сформировать в изображении тени, нужно математически описать физику процесса взаимодействия света и материалов реальных объектов. Данная задача сложна из-за объема вычислений, которые при отображении динамических объектов должны выпол-

няться в реальном масштабе времени. Однако без формирования теней нельзя обойтись при моделировании реалистичных изображений, потому в системах, где требуется реалистичность изображения и высокая динамика, приходится идти на различные хитрости.

Рассмотрим, как образуется тень при использовании единственного точечного источника света (рис. 2.11). Источник называется точечным, если свет от него падает равномерно во все направления. Для простоты предположим, что тень падает на плоскую поверхность $y = \text{const}$. При этом на плоскости $y = 0$ образуется треугольник тени, который является ничем иным, как проекцией объекта на заданную поверхность. Центр проецирования при этом располагается в точке, где находится источник света. Если сформировать проекцию исходного объекта — треугольника во фрейме, начало координат которого совпадает с источником света, то получим вершины многоугольника тени. Далее нужно сформировать представление полученных вершин в мировой системе координат. Эту задачу можно решать в прикладной программе, но гораздо рациональнее подобрать подходящую матрицу проективного преобразования и передать дальнейшие заботы о формировании многоугольника тени графической системе.

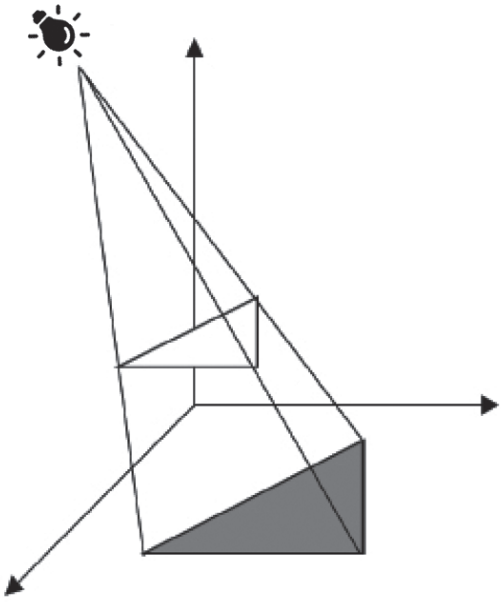

Рис. 2.11. Тень от треугольника

Предположим, что источник света находится в точке (x_l, y_l, z_l), как показано на рис. 2.11. С помощью матрицы сдвига $T(-x_l, -y_l, -z_l)$ можно

изменить положение объектов на рисунке таким образом, чтобы источник света оказался в центре систеемы координат (рис. 2.12). В результате получим обычную перспективную проекцию с центром проецирования в начале координат. Матрица проективного преобразования имеет вид

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & \dfrac{1}{-y_l} & 0 & 0 \end{bmatrix}.$$

Далее необходимо сдвинуть все обратно с помощью матрицы сдвига $T(-x_l, -y_l, -z_l)$. Суперпозиция двух сдвигов и проективного преобразования проецирует вершину (x, y, z) в точку (x_p, y_p, z_p) с координатами:

$$x_p = x_l - \frac{x - x_l}{(y - y_l)/y_l};$$

$$y_p = 0;$$

$$z_p = z_l - \frac{z - z_l}{(y - y_l)/y_l}.$$

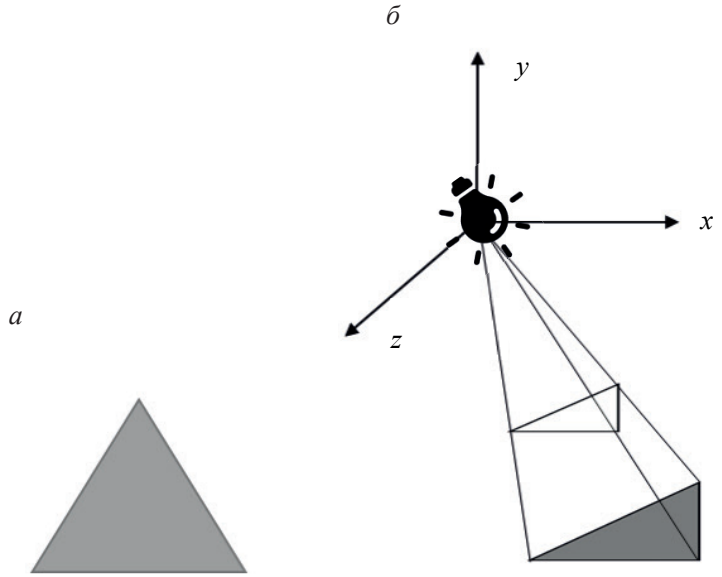

Рис. 2.12. Источник (*а*), который сдвинут в начало координат (*б*)

В графической системе необходимо задать матрицу проективного преобразования и выполнить умножение ее на матрицу вида. При этом треугольник вычерчивается дважды: первый раз как обычно, а второй раз с использованием измененной матрицы вида, которая преобразует вершины многоугольника. Для исходного многоугольника и для его тени используются одни и те же параметры визуализации.

Упражнения к главе 2

1. Приведите примеры проекций, где поверхность, на которую выполняется проецирование, является криволинейной. Придумайте примеры, в которых линии проецирования не являются прямыми.

2. Выведите матрицу перспективного проективного преобразования для случая, когда центр проецирования является произвольной точкой в пространстве, а картинная плоскость имеет произвольную ориентацию.

3. Докажите, что перспективное проективное преобразование сохраняет прямолинейность линий.

4. Любая попытка проецировать точку, которая принадлежит плоскости, параллельной картинной и проходящей через центр проецирования, приведет к делению на нуль. Какой вид будет иметь проекция прямолинейного отрезка, если одна из конечных его точек лежит на такой плоскости?

5. Косоугольное проецирование можно представить как первоначальный скос объектов с помощью матрицы скоса $H(\theta, \varphi)$, а затем выполнить обычное ортогональное проецирование. Найдите матрицу преобразования скоса.

6. Выведите матрицу косоугольного проективного преобразования.

7. Если для вычерчивания осей использовать ортогональное проецирование, то оси x и y будут лежать в плоскости чертежа, а ось z преобразуется в точку. Можно так организовать ортогональное проецирование, что проекции осей x и y будут по-прежнему пересекаться под углом 90°, а проекция оси z будет проходить под углом $-135°$ по отношению к оси x. Найдите соответствующую матрицу проективного преобразования.

Глава 3. Закрашивание

Во второй главе был затронут вопрос взаимодействия света и материала. Этот аспект формирования реалистичного компьютерного изображения весьма важен. В этой главе будут рассмотрены модели разных источников света и наиболее распространенных физических явлений, возникающих при взаимодействии света с поверхностями материальной среды. Целью главы является показать, как в уже описанную выше конвейерную архитектуру графической системы включить еще одну очень важную функцию — закрашивание объектов сцены с учетом характеристик источников света и материала моделируемых объектов. Особое внимание будет уделено локальным моделям распределения света. Такие модели, в противоположность глобальным моделям освещения, позволяют вычислить оттенок отдельной точки на некоторой поверхности, независимо от других поверхностей как этого, так и других объектов сцены. В процессе вычислений учитываются: характеристики материала, наложенного на поверхность, локальная геометрия этой поверхности, расположение и свойства источников света.

3.1. Свет и материя

С точки зрения физики поверхность материального тела может либо излучать световую энергию, например поверхность электрической лампочки, либо отражать свет, падающий на нее от внешнего источника. Некоторые тела одновременно и отражают свет, и излучают его вследствие внутренних физических процессов, происходящих в материале. Когда мы смотрим на некоторую точку материального объекта, то ее цвет определяется множеством элементарных актов взаимодействия со светом, падающим на объект как непосредственно от источников света, так и от других отражающих поверхностей. Последовательность этих элементарных актов можно представить в виде рекурсивного процесса. Рассмотрим простую сцену, представленную на рис. 3.1. Часть света от источника освещения, которая попадает на поверхность объекта А, отражается. Часть этого отраженного света попадает на поверхность объекта Б, при этом часть отраженного света отражается и попадает вновь на объект А. Далее процесс повторяется.

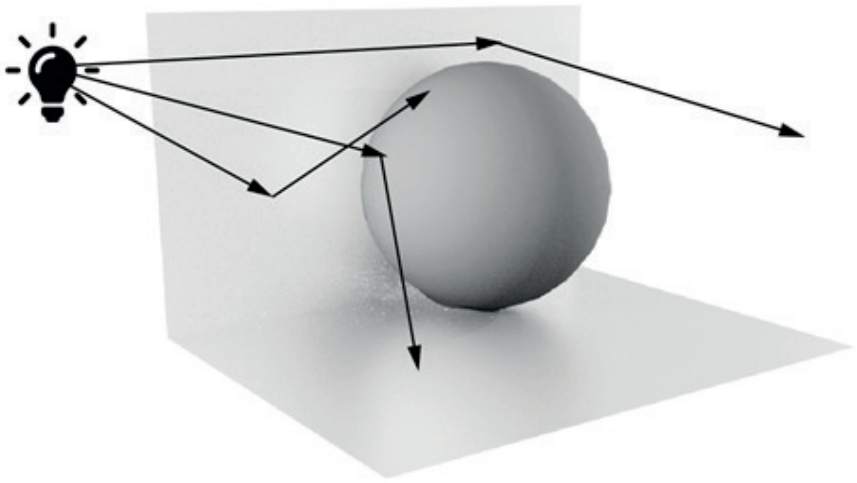

Рис. 3.1. Отражающие поверхности

Такое рекурсивное отражение света от двух поверхностей приводит к определенным цветовым эффектам, в частности к появлению на поверхностях дополнительных окрашенных бликов. Математически этот рекурсивный процесс описывается интегральным уравнением, которое называется *уравнением рендеринга*, или *уравнением глобального освещения*. Уравнение рендеринга определяет количество светового излучения в определенном направлении как сумму собственного и отраженного излучения. Существует множество подходов решения этого уравнения, например, метод трассировки лучей, метод излучательности, применение локальных оценок методом Монте-Карло [3]. Каждый из методов позволяет найти решение уравнения глобального освещения для определенного типа объектов. Поэтому основное внимание мы уделим более простым локальным моделям заполнения, основанным на модели отражения Фонга [11], которая представляет собой вполне приемлемый компромисс между физической корректностью и объемом необходимых вычислений.

Вместо того чтобы рассматривать уравнения общего баланса световой энергии, модель Фонга анализирует световые лучи, испускаемые светоизлучающими поверхностями — источниками света (*light sources*), и их взаимодействие с отражающими поверхностями объектов сцены. В чем-то такой подход сходен с трассировкой лучей, но, в отличие от последнего метода, он анализирует только один акт отраже-

ния от отдельной поверхности, а не рекурсивный процесс. Предлагается разделить проблему на две независимые части.

Во-первых, в модель сцены нужно включить описания источников света. Во-вторых, сформировать модель процесса отражения, которая будет адекватно передавать взаимодействие материала поверхностей объектов сцены и света.

Для того чтобы представить себе этот процесс, попробуем проследить, как распространяются лучи от точечного источника света к сцене, представленной на рис. 3.2. Наблюдатель видит только те лучи, которые отразились от поверхностей объектов и достигли его глаз, совершив довольно сложное «путешествие» в пространстве сцены и «зацепив» по пути не один объект. Если луч попал в глаз непосредственно от источника, то наблюдатель будет воспринимать цвет света, испускаемого этим источником. Если луч по пути отразился от поверхности какого-либо объекта, то наблюдатель видит цвет, зависящий от характера взаимодействия света, падающего на поверхность, и от материала поверхности.

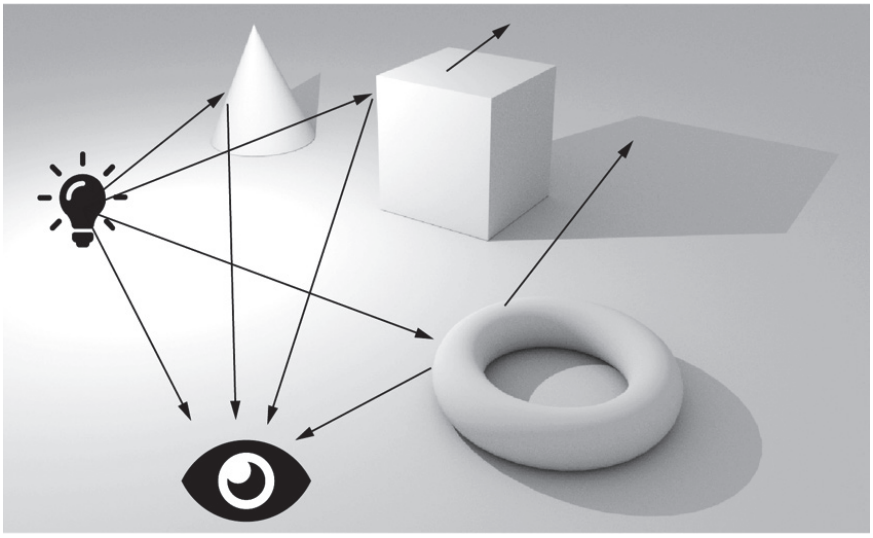

Рис. 3.2. Свет и поверхности

В компьютерной графической модели в роли глаза наблюдателя выступает картинная плоскость (рис. 3.3), а изображение формируют только те лучи, которые достигли центра проецирования, пройдя внутри зоны, ограниченной рамкой отсечения картинной плоскости.

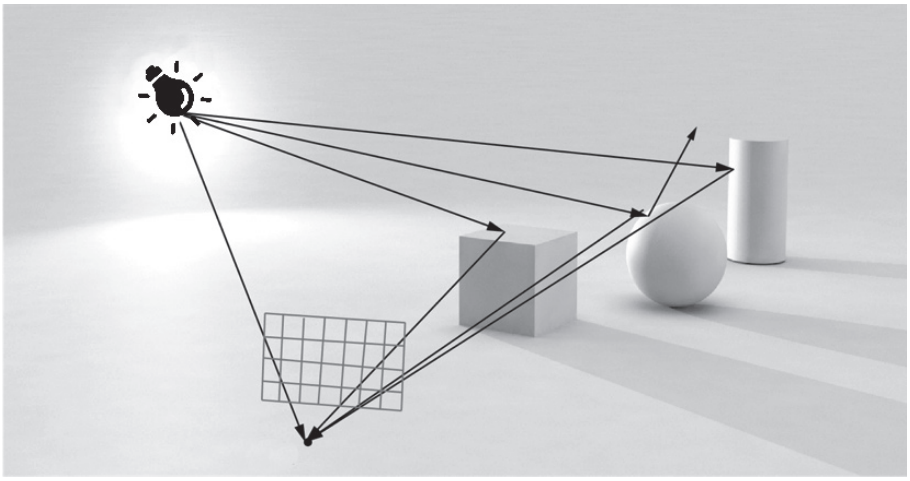

Рис. 3.3. Световые лучи, поверхности объектов и изображение на картинной плоскости

Большинство лучей, испускаемых источником, не попадает на картинную плоскость, а потому не представляет никакого интереса. Мы воспользуемся этим выводом позже. Будем считать, что картинная плоскость разделена на прямоугольники, каждому из которых соответствует отдельный пиксель экрана, причем цвет пикселя зависит от цвета луча, попавшего на этот пиксель.

На рис. 3.2 показаны варианты однократного и многократного взаимодействия луча с объектами. Именно характер этого взаимодействия и определяет, что же увидит наблюдатель: синий объект или фиолетовый, темный или светлый, матовый или блестящий. Когда луч попадает на поверхность, часть его энергии поглощается, а часть — отражается. Если поверхность непрозрачна, то вся световая энергия распределяется между процессами отражения и поглощения. Если же поверхность прозрачна, то часть энергии луча проходит через эту поверхность и может взаимодействовать с другими объектами.

Характер взаимодействия зависит от длины световой волны. Объект, освещенный белым светом, будет казаться наблюдателю красным, если большая часть энергии падающего света поглощена материалом объекта, а та, что отразилась, имеет максимум интенсивности в красной части спектра. Объект кажется наблюдателю блестящим, если его поверхность гладкая. И наоборот, объект, поверхность которого шероховата, кажется наблюдателю матовым.

Распределение теней на поверхности объекта зависит также от ориентации отдельных участков его поверхности, т. е. от направления вектора нормали в каждой точке этой поверхности.

Можно выделить три основных типа характера взаимодействия света и материала поверхности (рис. 3.4).

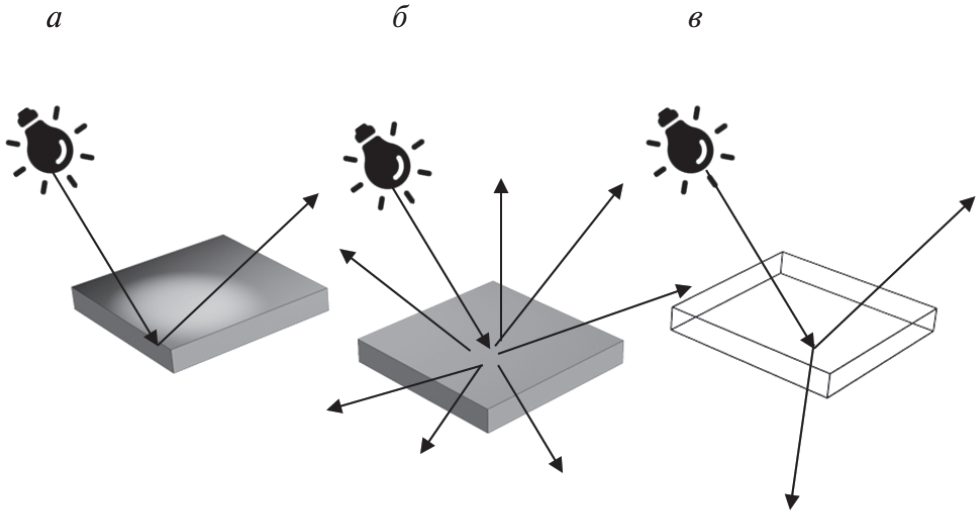

Рис. 3.4. Взаимодействие света и материала:
а — зеркальное отражение; *б* — диффузное отражение; *в* — преломление

При *диффузном отражении* падающий свет рассеивается от поверхности во всех направлениях. Такой тип взаимодействия характерен для большинства поверхности реального мира, например, равномерно окрашенной стены.

При *зеркальном отражении* небольшая часть энергии падающего луча поглощается, большая часть световой энергии отражается в узком диапазоне углов. Поверхности при зеркальном отражении выглядят блестящими, а под определенным углом виден световой блик. Зеркало — это идеально отражающая поверхность, весь падающий свет отразится от поверхности.

При *преломлении* луч света, падающий на поверхность, преломляется, проникает в среду объекта и выходит из нее под другим углом. Этот процесс характерен для стекла и воды. Как правило, при этом отражается только часть падающего света.

Источники света. Свет может исходить от поверхности объекта в двух случаях: либо объект излучает свет, либо отражает свет. Как правило, источником света мы считаем излучающие объекты, хотя это утверждение не совсем верно. Например, в ночных сценах источником может быть Луна, которая сама свет не излучает. Кроме того, некоторые объекты, которые излучают свет, одновременно и отражают свет, падающий на них от других источников, например, колба электрической лампочки.

Физический источник света имеет определенную поверхность (рис. 3.5). Каждая точка этой поверхности (*x, y, z*) может испускать световой луч, который характеризуется направлением эмиссии и распределением световой энергии по длинам волн. Таким образом, источник света в общем случае характеризуется функцией излучения (*illumination function*), зависящей от шести параметров: координаты источника света (*x, y, z*); длины волны (λ) и углов, описывающих направление излучения (θ, φ).

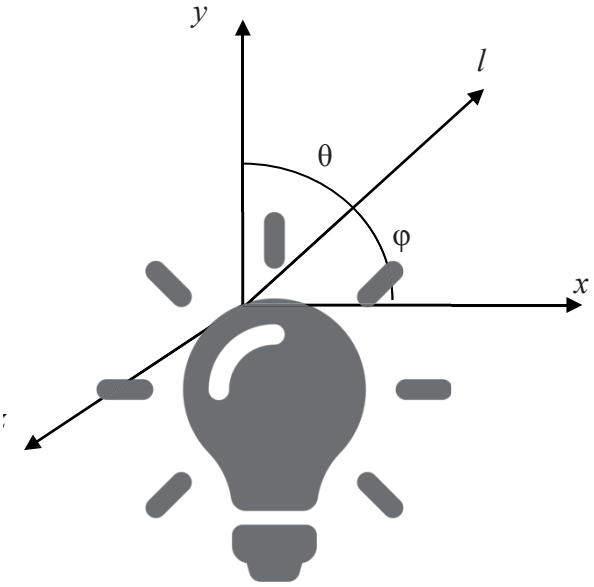

Рис. 3.5. Источник света

Кроме того, источник полихромного излучения представляется как множество независимых элементарных источников монохроматического света. Рассматривая освещаемую таким источником поверхность, можно получить освещенность каждой ее точки, интегрируя функцию излучения по поверхности источника света (рис. 3.6).

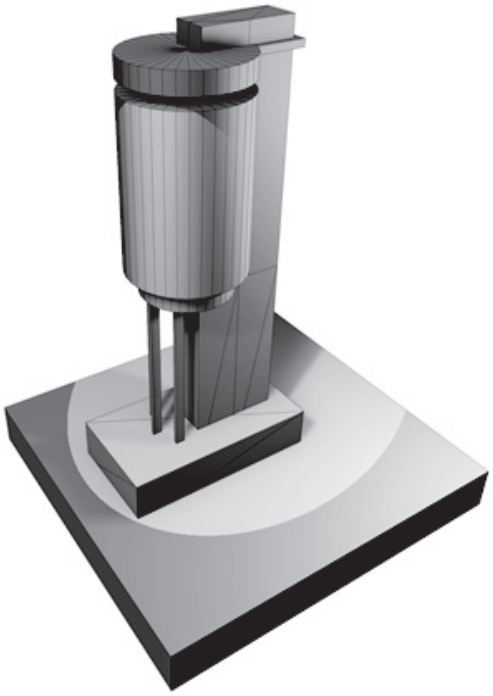

Рис. 3.6. Суммирование света от элементарных источников

При интегрировании нужно учитывать только те углы излучения, которые обеспечивают попадание лучей на анализируемую точку освещаемой поверхности, а также расстояние между элементарным источником и этой точкой (*d*). Для распределенного источника света, каковым является электрическая лампа, вычислить такой интеграл довольно сложно. Для упрощения подобный источник часто моделируется многоугольниками, каждый из которых представляет собой элементарный источник, или аппроксимацией реального источника множеством точечных источников.

В компьютерной графике для моделирования освещения используются четыре основных типа источников света: удаленные источники (*distant light*), точечные источники (*point light*), прожекторы (*spot light*) и фоновое освещение (*ambient lighting*).

Модель каждого источника света описывается трехкомпонентной функцией излучения в соответствии с трехцветной моделью RGB

$$I = \begin{bmatrix} I_r \\ I_g \\ I_b \end{bmatrix}.$$

Необходимо выяснить, как рассчитываются данные компоненты для основных типов источников света.

Согласно закону И. Ламберта яркость рассеивающей свет (диффузной) поверхности одинакова во всех направлениях. Освещенность некоторой точки поверхности светом от точечного источника обратно пропорциональна квадрату расстояния между этой точкой и источником (рис. 3.7). Освещенность в точке **p** описывается как

$$I(\mathbf{p}) = \frac{1}{d^2} I(\mathbf{p}_0).$$

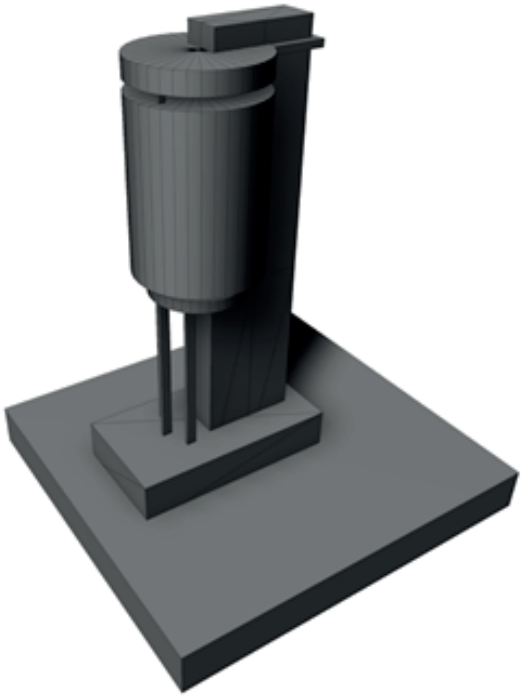

Рис. 3.7. Освещение поверхности точечным источником

Изображение сцены, в которой аппроксимируются только точечные источники света, получается очень контрастным: все объекты оказываются либо очень яркими, либо слишком темными. Для снижения контраста обратно пропорциональную зависимость между освещенностью и расстоянием заменяют полиномом $\left(ad^2 + bd + c\right)^{-1}$. Константы a, b, c выбирают эмпирическим путем. В компьютерной системе по умолчанию заданы $a = 1$, $b = 0$, $c = 0$.

Кроме того, снизить контраст от точечного источника можно с помощью источника фонового света. Такой источник освещает все объ-

екты сцены с одинаковой интенсивностью. Значит, каждый из компонентов функции I_a (красный, зеленый, синий) является скаляром. Однако необходимо учитывать, что разные поверхности по-разному отражают свет одинаковой интенсивности.

Удаленный источник света характеризуется тем, что все испускаемые им лучи можно считать параллельными (рис. 3.8).

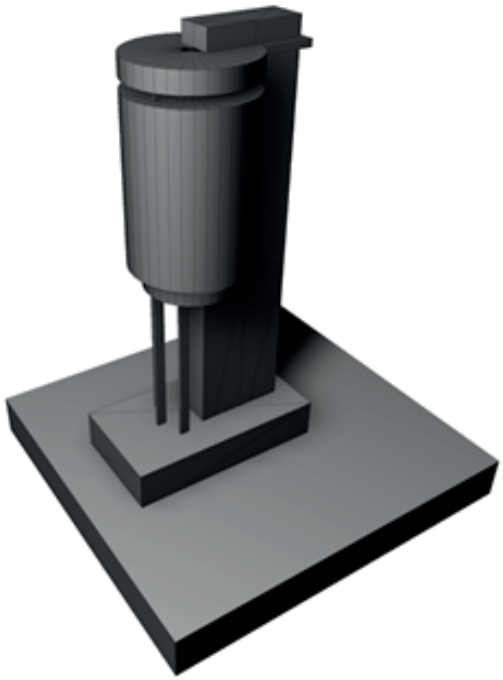

Рис. 3.8. Освещение поверхности удаленным источником

Использование такого источника избавляет от необходимости рассчитывать направление лучей, освещающих разные точки отображаемой поверхности, что в результате существенно повышает скорость формирования изображения. Математически обработка удаленного источника выполняется аналогично параллельному проецированию. Вместо положения источника света учитывается направление его лучей. Если использовать математический аппарат однородных координат, то удаленный источник света представляется вектором направления

$$\mathbf{p}_0 = \begin{bmatrix} x \\ y \\ z \\ 0 \end{bmatrix}.$$

Точечный источник света описывается в виде

$$\mathbf{p}_0 = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$$

Источник света типа прожектора испускает свет в одном направлении, как свет от фонарика (рис. 3.9). Такой источник моделируют с помощью точечного источника, ограничив направление распространения световых лучей. Наибольшей интенсивностью обладают лучи, направленные вдоль оси конуса. Интенсивность излучения прожектора является функцией от угла φ между осью конуса излучения и вектором *s*, направленным на точку освещаемой поверхности,

$$I(\mathbf{p}) = I(\mathbf{p}_s)\cos^c\varphi\ ,$$

где *c* — показатель, определяющий, насколько быстро убывает интенсивность.

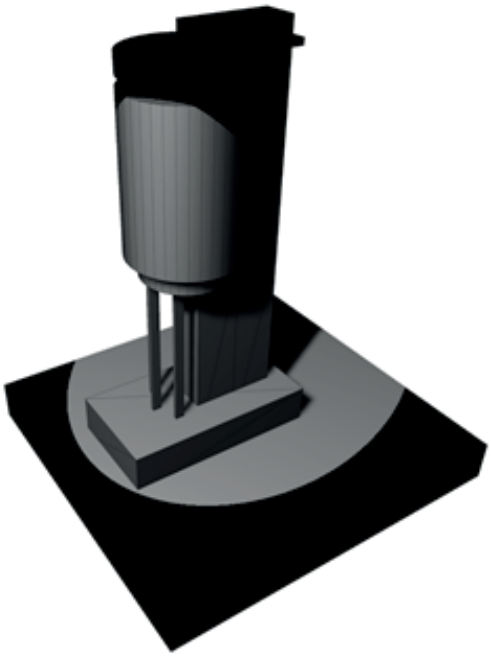

Рис. 3.9. Освещение поверхности прожектором

3.3. Модель отражения Фонга

Компьютерное моделирование графики столкнулось с довольно сложной задачей: с одной стороны, нужна модель взаимодействия света и материала, которая бы соответствовала реальным физическим процессам, а с другой стороны, реализация этой модели в графической системе должна хорошо встраиваться в конвейерную архитектуру и не слишком перегружать вычислительную систему компьютера. Аппроксимации Фонга, или затенение по Фонгу (*Phong shading*), носят название по имени своего изобретателя — Ву Тонг Фонга (*Wu Tong Phong*). Далее будет рассмотрена модель процесса отражения света объектами сцены, предложенная Фонгом.

Успешная практика использования этой модели во множестве графических систем доказала ее эффективность. Модель позволяет формировать полутоновые изображения высокого качества при самых различных условиях освещения и свойствах материалов. Модель Фонга адекватно передает три из четырех типов взаимодействия света и материала: зеркальное, диффузное и фоновое освещение.

Для вычисления цвета произвольной точки **p** на поверхности объекта используется четыре вектора (рис. 3.10). Вектор *n* является нормалью к поверхности в точке **p**. Вектор *v* направлен от точки **p** к наблюдателю или центру проецирования, а вектор *l* задает направление от точки **p** к произвольной точке источника света. Вектор *r* — это направление идеального отражения луча, падающего на поверхность вдоль вектора *l*. Вектор *r* однозначно определяется векторам *n* и *l*.

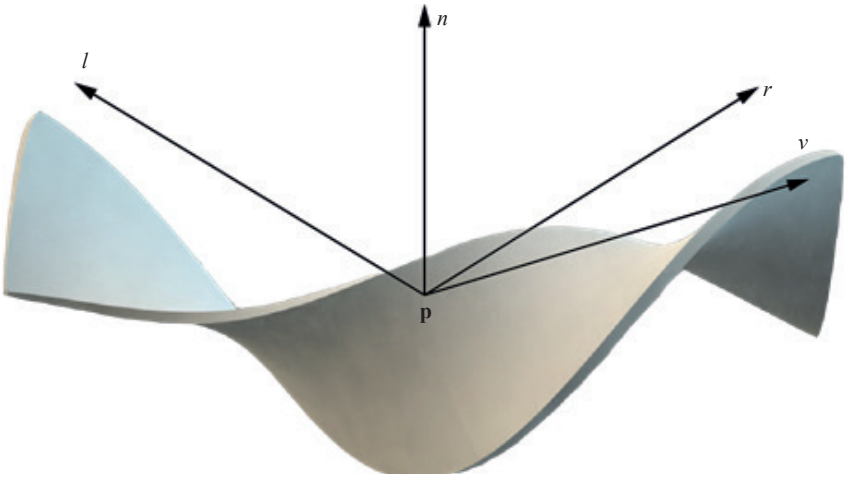

Рис. 3.10. Векторы в модели Фонга

Когда моделируемая поверхность криволинейная (наиболее распространенный случай в реальном мире), все четыре вектора могут изменяться при переходе от одной точки к другой. Суть аппроксимаций заключается в определении нормали к поверхности в каждой вершине полигона и дальнейшей интерполяции вектора по всему полигону поверхности. Далее для каждого пикселя необходимо вычислить значение яркости, основываясь на значениях вектора нормали.

Предположим, что имеется множество точечных источников света. Можно считать, что для каждого из трех основных цветов свет от этого источника, влияющий на формируемое изображение, имеет три составляющие, которые назовем компонентами фонового (*ambient*), зеркального (*specular*) и диффузного (*diffuse*) света. (Соответственно в математических выражениях компонент фонового цвета будет иметь индекс *a*, зеркального — индекс *r*, а диффузного — *d*.) Для каждой точки **p** отображаемой поверхности можно вычислить матрицу освещенности размером 3×3 для *i*-го источника света:

$$L_i = \begin{bmatrix} L_{ira} & L_{iga} & L_{iba} \\ L_{ird} & L_{igd} & L_{ibd} \\ L_{irs} & L_{igs} & L_{ibs} \end{bmatrix}.$$

Элементы первой строки матрицы представляют интенсивности фонового света соответственно для красного, зеленого и синего цветов, поступающего от *i*-го источника. Элементы второй строки представляют цветовые компоненты интенсивности для диффузного света, а третьей — для зеркального. Предполагается, что в этих элементах не учитываются эффекты ослабления интенсивности освещения, связанные с расстоянием до источника.

Предположим, что существуют методы вычисления коэффициентов отражения для всех компонентов матрицы, тогда, зная значение диффузной составляющей, например, для красного цвета L_{ird}, поступающего от *i*-го источника, и зная коэффициент отражения R_{ird} для соответствующего компонента, можно вычислить значение составляющей интенсивности света, отраженного точкой **p**, в виде произведения $R_{ird}L_{ird}$. Значение коэффициента отражения R_{ird} зависит от свойств материала поверхности, ее ориентации, направления на источник света и расстояния между источником и освещаемой точкой. Для каждой точки можно вычислить свой набор коэффициентов:

$$R_i = \begin{bmatrix} R_{ira} & R_{iga} & R_{iba} \\ R_{ird} & R_{igd} & R_{ibd} \\ R_{irs} & R_{igs} & R_{ibs} \end{bmatrix}.$$

Сумма интенсивностей отраженного света для диффузной, фоновой и зеркальной составляющих и будет определять интенсивность цветовой компоненты для *i*-го источника. Например, интенсивность составляющей красного цвета *i*-го источника, отраженной от поверхности в точке **p**,

$$I_{ir} = R_{ira} L_{ira} + R_{ird} L_{ird} + R_{irs} L_{irs}.$$

Интегральную интенсивность отраженного света несложно найти, просуммировав составляющие от всех источников, имеющихся в сцене:

$$I_r = \sum_i \left(I_{ira} + I_{ird} + I_{irs} \right) + I_{ar}.$$

где I_{ar} обозначает красную составляющую глобального фонового освещения.

Упростим вид выражений, опустив индекс цвета, поскольку все сформулированные соотношения справедливы для каждого из трех основных цветов. Вычисление коэффициентов отражения для составляющих фонового, диффузного и зеркального света будет существенно отличаться. Отбросив индексы цвета, получим приведенное выше выражение в более компактном виде:

$$I = I_a + I_d + I_s = R_a L_a + R_d L_d + R_s L_s .$$

Вычисления по приведенной формуле нужно выполнить для каждого основного цвета и каждого из имеющихся в сцене источников, затем просуммировать результаты по всем источникам и прибавить к ним интенсивность глобального фонового освещения.

3.4. Модель направленного света

На основе простой модели освещения с точечным источником можно рассмотреть модель освещения, включающую специальные эффекты, которые применяются при управлении светом в профессиональных фотостудиях. К ним относятся задание направления и концентрации света, а также возможность ограничить область, освещаемую источником света (конус). Направление света можно регулировать независимо от расположения источника (рис. 3.11). Теоретически направленный

свет можно представить как отражение света точечного источника от идеальной псевдоповерхности (рис. 3.11). Если точечный источник находится в направлении, перпендикулярном отражающей псевдоповерхности, то отражение исходящего от него света освещает объект в этом направлении. Направление света регулируется поворотом псевдоповерхности.

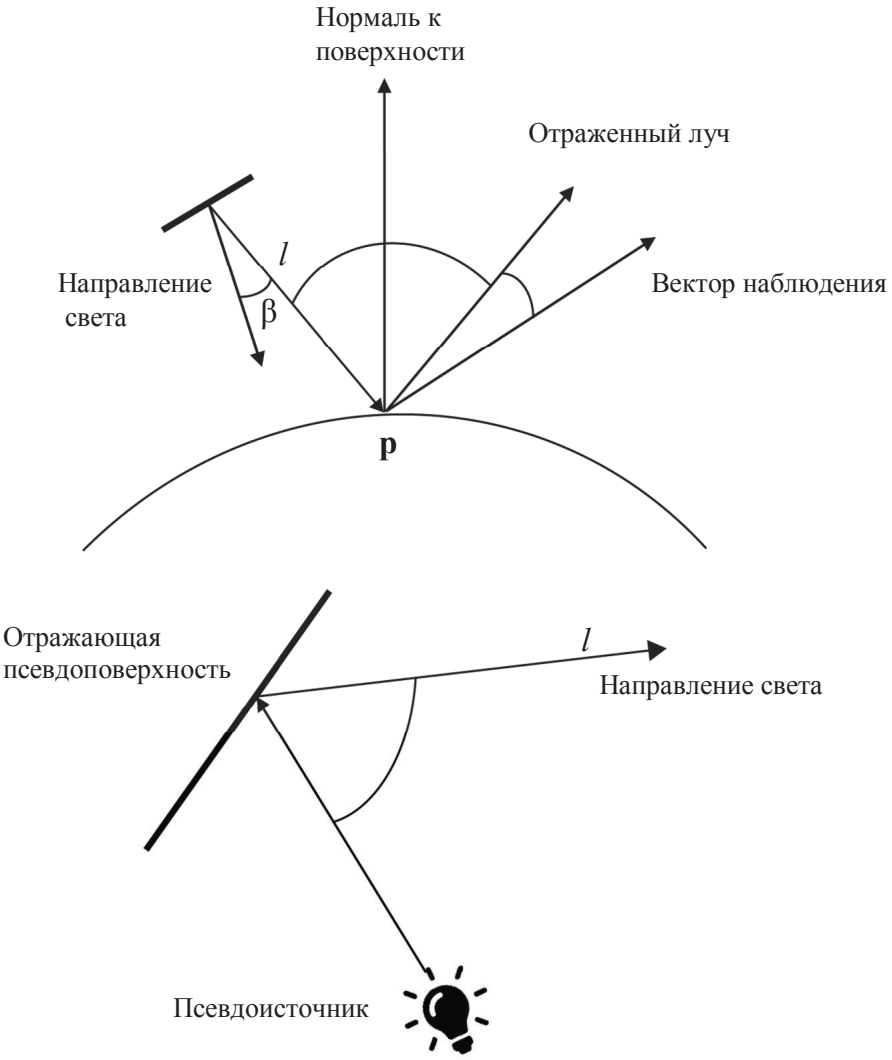

Рис. 3.11. Модель направленного освещения

При таком подходе одну и ту же модель освещения можно применить как для направленных, так и для точечных источников света.

Количество света, падающего в точку **p** от направленного источника, зависит, как показано на рис. 3.11, от угла между вектором направления света ***l*** и прямой, проходящей через всевдоисточник и точку отражающей всевдоповерхности. Пользуясь аппроксимацией Фонга для зеркального отражения от идеальной поверхности, находим интенсивность света от направленного источника вдоль прямой, соединяющей источник и точку **p**,

$$I(\mathbf{p}) = I_{l_d} \cos^c \beta,$$

где *c* — степень, определяющая пространственную концентрацию направленного источника. При большом *c* моделируется узкий луч прожектора, а при малом *c* — заливающий свет. Теперь интенсивность света от направленного источника в модели освещения будет

$$I_j = I_{l_j} \cos^c \beta \left(k_{d_j} \cos\theta_j + k_{s_j} \cos^{n_j} \alpha_j \right),$$

где *j* обозначает конкретный источник света.

В фотостудиях достигают специальных эффектов с помощью ограничения освещаемых участков заслонками, которые устанавливаются на источниках света, а также с помощью особых отражателей. Для создания специальных эффектов можно применять заслонки и конусы.

Заслонки, ориентированные по координатным плоскостям, моделируются путем ограничения протяженности освещенного участка по осям *x, y* и *z* (рис. 3.12, *а*). Если точка объекта лежит внутри ограничителей, например $y_{\min} \leqslant y_{\text{объект}} \leqslant y_{\max}$, то вычисляется компонента интенсивности, определяемая этим источником, в противном случае она игнорируется. Используя заслонки, можно создавать эффекты, не встречающиеся в природе, например, можно закрыть часть сцены от некоторого источника света.

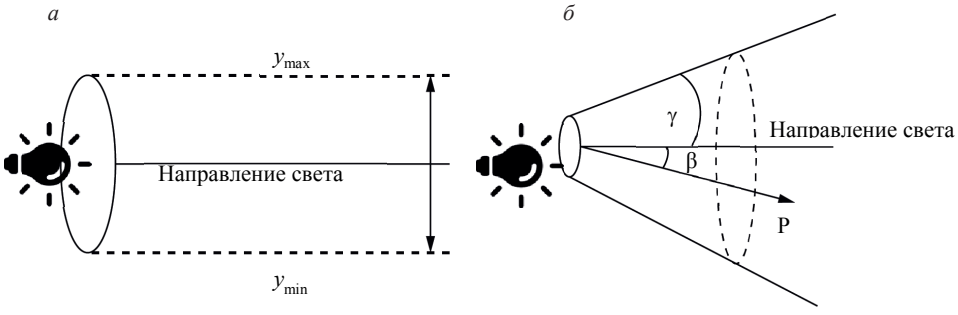

Рис. 3.12. Заслонка (*а*) и воронка (*б*)

Конус (рис. 3.12, *б*) позволяет получить четко очерченное пятно света в отличие от направленного источника, при котором свет постепенно затухает у края. Этот эффект также заимствован из фотографии. Пусть вершина конуса совпадает с источником, а Υ — угол раствора конуса, тогда при $\beta > \Upsilon$ точка **p** не освещается этим источником. В противном случае соответствующая компонента интенсивности включается в модель освещения. Обычно для этого сравниваются $\cos\beta$ и $\cos\Upsilon$, т. е. должно быть $\cos\beta > \cos\Upsilon$.

3.5. Прозрачность объекта

В основных моделях освещения и алгоритмах удаления невидимых линий и поверхностей рассматриваются только непрозрачные поверхности и объекты. Однако существуют и прозрачные объекты, пропускающие свет, например такие, как стакан, ваза, окно автомобиля, вода. При переходе из одной среды в другую, например из воздуха в воду, световой луч преломляется, поэтому торчащая из воды палка кажется согнутой. Преломление рассчитывается по закону Снеллиуса, который утверждает, что падающий и преломляющий лучи лежат в одной плоскости, а углы падения и преломления связаны по формуле

$$\eta_1 \sin\theta = \eta_2 \sin\theta',$$

где η_1 и η_2 — показатели преломления двух сред; θ — угол падения; θ' — угол преломления (рис. 3.13). Ни одно вещество не пропускает весь падающий свет, часть его всегда отражается, т. е. существует отраженный луч.

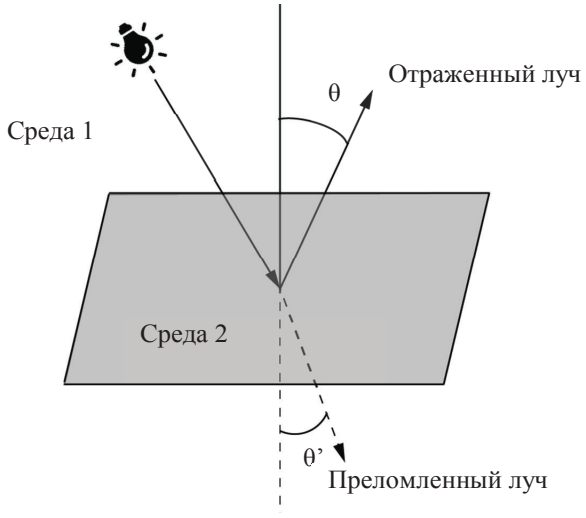

Рис. 3.13. Геометрия преломления

Так же как и отражение, пропускание может быть зеркальным (направленным) или диффузным. Направленное пропускание свойственно прозрачным веществам, например стеклу. Если смотреть на объект сквозь такое вещество, то, за исключением контурных линий криволинейных поверхностей, искажения происходить не будет. Если свет при пропускании через вещество рассеивается, то мы имеем диффузное пропускание. Такие вещества кажутся полупрозрачными или матовыми. Если смотреть на объект сквозь подобное вещество, то он будет выглядеть нечетким или искаженным. На рис. 3.14 показаны некоторые следствия преломления. Показатель преломления объектов 1 и 2 одинаков и больше, чем у окружающей среды. Объекты 3 и 4 непрозрачны. Если не принимать во внимание преломление, то луч *a* пересечется с объектом 4 (пунктирная линия). Однако из-за преломления он отклоняется и пересекается с объектом 3, т. е. объект 3 виден только благодаря эффекту преломления. Если же рассматривать луч *b*, то без учета преломления он пересекается с объектом 3, хотя на самом деле он пересекается с объектом 4, т. е. здесь объект, который видим, на самом деле увидеть нельзя. Все это необходимо иметь в виду при создании реалистических изображений.

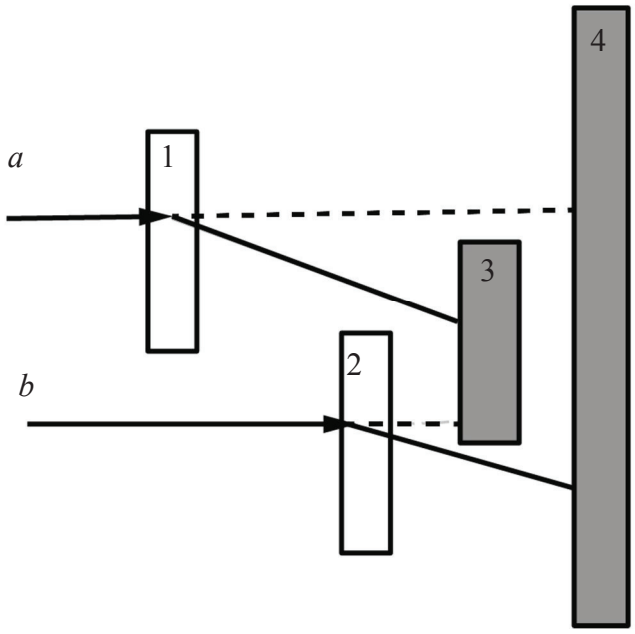

Рис. 3.14. Эффекты преломления:
1–4 — объекты преломления

Нечто похожее происходит после перспективного преобразования, которое искажает объект для того, чтобы затем выполнить ортогональное проецирование. На рис. 3.15, *а* луч, исходящий из точки **p**, пересекает неискаженный объект в точке *i* и после преломления попадает в точку *b* плоскости фона. На рис. 3.15, *б* показан объект после перспективного преобразования.

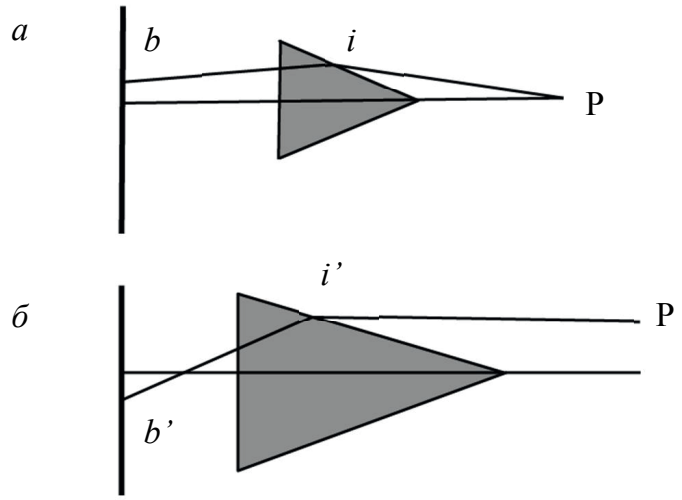

Рис. 3.15. Влияние перспективы на преломление

Теперь луч пересекает объект в преобразованной точке *i'*, преломленный луч пересекается с фоном в точке *b'* с *противоположной* стороны от центральной линии. Это происходит из-за неправильных угловых соотношений между искаженным (преобразованным) объектом и искаженным (преломленным) лучом.

На первый взгляд, для получения верного результата достаточно знать истинные угловые соотношения в точках пересечения луча с объектом. Однако это не так, потому что длина пути луча в преобразованном объекте также меняется. Разница в длине пути приводит к тому, что, во-первых, не совпадают точки выхода луча из объекта, поэтому луч все равно не попадает в правильную точку фона. Во-вторых, меняется количество поглощенного объектом света, поэтому исходящий луч имеет другую интенсивность.

Для того чтобы устранить влияние преломления, можно либо применять алгоритмы, работающие в пространстве объекта, либо пользоваться специальными преобразованиями между пространствами объ-

екта и проекцией. Однако проще включить преломление в алгоритмы удаления невидимых поверхностей.

В простейших реализациях компьютерных изображений эффекты прозрачности преломления вообще не рассматриваются и явления, показанные на рис. 3.14 и 3.15, не учитываются. Кроме того, не принимается во внимание, как путь, пройденный лучом в среде, влияет на его интенсивность.

Простое пропускание света можно встроить в любой алгоритм удаления невидимых поверхностей, кроме алгоритма z-буфера. Прозрачные многоугольники или поверхности помечаются, и если видимая грань прозрачна, то в буфер кадра записывается линейная комбинация двух ближайших поверхностей. При этом интенсивность

$$I = tI_1 + (1-t)I_2,\ 0 \leqslant t \leqslant 1,$$

где t — коэффициент прозрачности для первой поверхности; I_1 — интенсивность света для видимой поверхности; I_2 — интенсивность света для поверхности, расположенной непосредственно за ней.

Если поверхность совершенно прозрачна, то $t = 0$, а если непрозрачна, то $t = 1$. Если вторая поверхность тоже прозрачна, то алгоритм применяется рекурсивно, пока не встретится непрозрачная поверхность или фон. Если многоугольники записываются в буфер кадра в соответствии с приоритетами глубины, как в алгоритме Ньюэла — Ньюэла — Санча[5], то I_2 будет соответствовать значению, записанному в буфер кадра, а I_1 — текущей поверхности.

Для криволинейных поверхностей, например таких, как ваза или бутылка, линейной аппроксимации недостаточно, поскольку вблизи контурных линий прозрачность уменьшается из-за толщины материала (рис. 3.16). Чтобы точнее изобразить это явление, необходимо провести несложную нелинейную аппроксимацию на основе z-составляющей нормали к поверхности, в частности, коэффициент прозрачности

$$t = t_{\min} + (t_{\max} - t_{\min})[1 - \left(1 - |n_z|^p\right],$$

где t — прозрачность точки объекта; $t_{\min}$ и $t_{\max}$ — минимальная и максимальная прозрачность объекта; n_z — z-составляющая единичной нормали к поверхности; p — коэффициент степени прозрачности.

[5] Метод сортировки, при котором перед обработкой каждого кадра динамически вычисляется новый список приоритетов сцены.

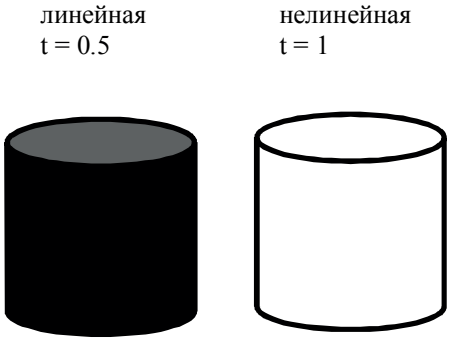

Рис. 3.16. Сравнение простых моделей прозрачности

Непосредственно в алгоритм, использующий z-буфер, эффект прозрачности ввести нельзя. Для того чтобы его учесть, необходимо использовать отдельные буферы прозрачности, интенсивности и весовых коэффициентов, а также нужно отметить прозрачные многоугольники в структуре данных. Для алгоритма, использующего z-буфер, последовательность действий такова:

1) для каждого многоугольника — если многоугольник прозрачен, то внести его в список прозрачных многоугольников; если многоугольник непрозрачен и координата $z > z_{\text{буфер}}$, то записать его в буфер кадра для непрозрачных многоугольников и скорректировать этот буфер;

2) для каждого многоугольника из списка прозрачных многоугольников — если координата $z > z_{\text{буфер}}$, то прибавить его коэффициент прозрачности к значению, содержащемуся в буфере весовых коэффициентов прозрачности;

3) прибавить его интенсивность к значению, содержащемуся в буфере интенсивности прозрачности, в соответствии с правилом

$$I_{bn} = I_{bo}t_{bo} + I_c t_c,$$

где I_{bn} — новое значение интенсивности; I_{bo} — старое значение интенсивности, записанное в буфере интенсивности прозрачности; t_{bo} — старый коэффициент прозрачности из буфера весовых коэффициентов прозрачности; I_c — интенсивность многоугольника; t_c — коэффициент его прозрачности.

Таким образом получается взвешенная сумма интенсивностей всех прозрачных многоугольников, находящихся перед ближайшим непрозрачным многоугольником.

Далее объединим буферы интенсивности для прозрачных и непрозрачных многоугольников в соответствии с правилом

$$I_{fb} = t_{bo} I_{bo} + (1 - t_{bo}) I_{fbo},$$

где I_{fb} — окончательная интенсивность в буфере кадра для непрозрачных многоугольников, а I_{fbo} — старое значение интенсивности в этом буфере.

Одной из интересных прикладных задач, связанных с прозрачностью, является визуализация внутреннего вида сложных объектов или пространств. Для этого всем многоугольникам поверхности присваиваются коэффициенты прозрачности, первоначально равные 1, т. е. они считаются непрозрачными. Затем коэффициенты прозрачности некоторых групп граней заменяются на 0, т. е. они делаются невидимыми. При новом построении изображения сцены получается внутренний вид этого объекта или пространства.

Упражнения к главе 3

1. При анализе модели отражения Фонга не рассматривались источники света, лучи от которых по пути к анализируемой поверхности «наталкиваются» на другую поверхность. Объясните, почему такие источники игнорируются в этой модели.
2. Как повлияет на вычисления, связанные с тонированием изображения сцены, учет расстояния между отображаемой поверхностью и наблюдателем?
3. По мере того как геометрическая информация проходит по «конвейеру», она подвергается преобразованиям поворота, сдвига, масштабирования и проецирования. Такое же преобразование выполняется и с векторами, которые определяют косинусные составляющие в модели отражения Фонга. Сохраняет ли какое-либо из этих преобразований углы между векторами? Как повлияет сформулированный ответ на реализацию процедуры закрашивания?
4. Обобщите алгоритм формирования теней, описанный в гл. 2, на обработку плоских поверхностей, произвольно ориентированных в пространстве.
5. Модифицируйте алгоритм формирования теней таким образом, чтобы он мог работать с удаленным источником света. *Указание*: вместо перспективного проецирования используйте параллельное.

Глава 4. Вычисление векторов

Описанная в предыдущей главе модель отражения света никак не связана с видом проецирования (параллельным или перспективным). Она является общей, т. е. может применяться как к плоским, так и к криволинейным поверхностям, причем расстояние между поверхностью и наблюдателем не учитывается. Большая часть вычислений в процессе закрашивания изображений пространственной сцены приходится на определение компонентов векторов, используемых в модели Фонга, и их скалярных произведений. В процессе выполнения этих вычислений часто используются разнообразные упрощения, учитывающие специфику конкретной ситуации. Например, если анализируемая поверхность — плоский многоугольник, то для всех ее точек вектор нормали будет одним и тем же. Если источник света достаточно далеко удален от поверхности, то для всех точек этой поверхности вектор, характеризующий направление падения лучей от этого источника, будет одним и тем же.

В этой главе рассматриваются методы вычисления компонентов векторов для общего случая.

4.1. Вектор нормали к поверхности

Вектор нормали на гладкой поверхности существует в каждой ее точке и определяет локальную ориентацию участка поверхности в окрестности этой точки. Метод вычисления компонентов вектора нормали зависит от способа математического описания поверхности. Продемонстрируем, как вычислить нормаль для плоскости и сферы.

Плоскость описывается уравнением

$$ax + by + cz = 0.$$

Уравнение плоскости можно записать в виде произведения вектора нормали n в точке P_0 и любого вектора, принадлежащего этой плоскости,

$$\mathbf{n} \cdot \left(P - P_0 \right) = 0,$$

где P — любая точка (x, y, z) на рассматриваемой плоскости.

Плоскость может быть однозначно задана совокупностью трех точек плоскости $P_0, P_1, P_2,$ не лежащих на одной прямой (неколлинеар-

ных). Для определения нормали можно использовать их векторное произведение

$$\mathbf{n} = (P_2 - P_0) \times (P_1 - P_0).$$

Для любой поверхности в системе компьютерной графики различают внешнюю и внутреннюю стороны. От порядка сомножителей в векторном произведении зависит направление вектора нормали, а следовательно, определение внешней стороны поверхности. Некоторые графические системы автоматически вычисляют вектор нормали, используя первые три вершины в определении многоугольника, полагая его плоским. В библиотеки *OpenGL* этого не предусмотрено, но существует возможность самостоятельно организовать в программе определение нормали, что позволяет прикладному программисту гибко управлять свойствами модели освещения.

Способ вычисления компонентов вектора нормали к криволинейной поверхности существенно зависит от принятого способа описания поверхности. Покажем несколько способов определения нормали к сферической поверхности единичного радиуса, центр которой совпадает с началом координат. Обычно такая сфера описывается уравнением в неявной форме

$$x^2 + y^2 + z^2 - 1 = 0,$$

или в векторной форме

$$\boldsymbol{P} \cdot \boldsymbol{P} - 1 = 0.$$

Нормаль определяется *вектором градиента* (*gradient vector*), который представляется в виде матрицы-столбца

$$\boldsymbol{n} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \\ \dfrac{\partial f}{\partial z} \end{bmatrix} = \begin{bmatrix} 2x \\ 2y \\ 2z \end{bmatrix} = 2P.$$

Сфера может быть описана и уравнением в *параметрической форме*. При этом координаты *x, y* и *z* любой точки на сферической поверхности представляют собой независимые уравнения от двух параметров *u* и *v*

$$\begin{cases} x = x(u,v); \\ y = y(u,v); \\ z = z(u,v). \end{cases}$$

В системах компьютерной графики предпочтение следует отдать параметрической форме, особенно при описании кривых и поверхностей. Один из вариантов описания сферы имеет вид

$$\begin{cases} x = \cos u \sin v; \\ y = \cos u \cos v; \\ z = \sin u. \end{cases}$$

Изменяя u и v в диапазоне $-\pi/2 < u < \pi/2$, — $\pi/2 < v < \pi/2$, получим все точки на сфере.

Используя параметрическое представление поверхности, нормаль можно вычислить как характеристику касательной плоскости в точке $P(u, v)$ (рис. 4.1).

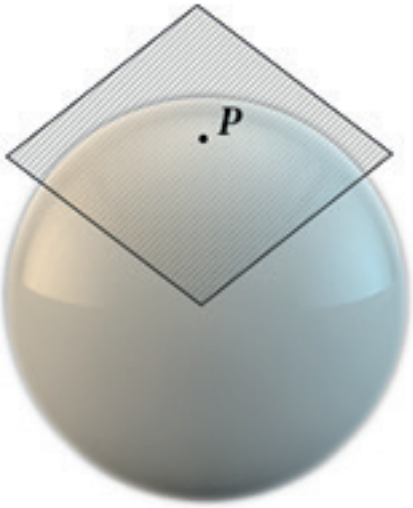

Рис. 4.1. Касательная плоскость к сферической поверхности

Получить уравнение касательной плоскости можно используя разложения параметрических функций сферы в ряд Тейлора в окрестности точки P. Прямые, параллельные векторам $\frac{\partial P}{\partial u}$ и $\frac{\partial P}{\partial v}$, проходят че-

рез точку P и лежат в касательной плоскости. Векторы $\frac{\partial P}{\partial u}$ и $\frac{\partial P}{\partial v}$ определяются следующим образом:

$$\frac{\partial P}{\partial u}=\begin{bmatrix}\frac{\partial x}{\partial u}\\ \frac{\partial y}{\partial u}\\ \frac{\partial z}{\partial u}\end{bmatrix},\quad \frac{\partial P}{\partial v}=\begin{bmatrix}\frac{\partial x}{\partial v}\\ \frac{\partial y}{\partial v}\\ \frac{\partial z}{\partial v}\end{bmatrix}.$$

Вектор нормали формируется как векторное произведение

$$n=\frac{\partial P}{\partial u}\times\frac{\partial P}{\partial v}.$$

Подставляя в это уравнение параметрические функции сферической поверхности, получим

$$n=\cos u\begin{bmatrix}\cos u\sin v\\ \cos u\cos v\\ \sin u\end{bmatrix}=(\cos u)\,P(u,v).$$

Поскольку нас интересует только направление вектора нормали, то для сферы единичного радиуса получим $\mathbf{n}=P$.

В графической системе основным типом примитива, с которым выполняются все вычисления, являются вершины. Таким образом, и вектор нормали следует формировать достаточно близко к той точке, нормаль в которой нас интересует. В рамках конвейерной архитектуры графических систем организовать такое вычисление довольно сложно, поскольку по «конвейеру» точки следуют одна за другой. Поэтому в большинстве графических систем программисту приходится самостоятельно организовывать вычисление векторов нормалей в прикладной программе.

4.2. Вектор отражения

Определив нормаль к поверхности в анализируемой точке и зная положение источника света, можно вычислить направление идеально отраженного луча. Для идеального зеркала выполняется закон: угол падения равен углу отражения. Угол падения измеряется между нор-

малью и направлением на источник света, а угол отражения измеряется между нормалью и отраженным лучом (рис. 4.2).

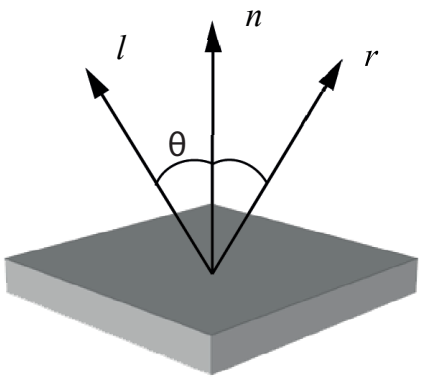

Рис. 4.2. Отражение от идеального зеркала

В двумерном пространстве направление отраженного луча задается этим законом однозначно, а в трехмерном пространстве для вычисления направления соответствующего вектора нужно ввести дополнительное условие: в точке *P* падающий и отраженный лучи, а также нормаль к поверхности должны лежать в одной плоскости, т. е. быть компланарными векторами. Это условие позволяет однозначно определить направление отраженного луча *r* по заданным векторам нормали **n** и падающего луча *l*. Поскольку нас интересует только направление отраженного луча *r*, то в дальнейшем будем предполагать, что все интересующие нас векторы являются ортами — векторами единичной длины.

Если $\theta_l = \theta_r$, то $\cos\theta_l = \cos\theta_r$. Используя скалярное произведение, получим соотношение, связывающее углы падения и отражения

$$\cos\theta_l = l \cdot n = \cos\theta_r = r \cdot n.$$

Условие компланарности трех векторов означает, что *r* можно выразить линейной комбинацией *l* и *n*

$$r = \alpha l + \beta n.$$

Правую и левую часть уравнения умножим скалярно на *n,* получим

$$n \cdot r = \alpha l \cdot n + \beta = l \cdot n.$$

Второе соотношение, связывающее параметры α и β, можно найти из условия, что вектор *r* должен быть единичным

$$l = r \cdot r = \alpha^2 + 2\alpha\beta l \cdot n + \beta^2.$$

Решая уравнение, получим, что $r = 2(l \cdot n)n - 1$.

4.3. Вектор половинного направления

Для учета зеркального отражения света на основе модели Фонга скалярные произведения $r \cdot v$ нужно вычислять для каждой точки поверхностей объектов сцены (рис. 4.3). Вектор *v* определяет направление наблюдателя. Можно упростить этот процесс, вычисляя промежуточный вектор единичной длиной *h,*

$$h = \frac{l + v}{|l - v|}.$$

Угол между векторами *n* и *h* называется углом половинного направления (*half-angle*). Если вектор *v* лежит в той же плоскости, что и *l*, *n* и *r,* то выполняется соотношение $2\psi = \varphi$.

Можно избавиться от вычисления вектора *r*, если заменить вычисление скалярного произведения $r \cdot v$ произведением $n \cdot h$.

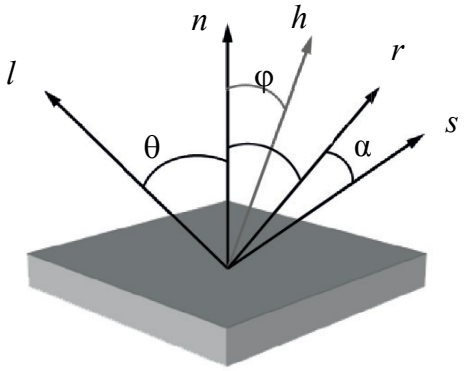

Рис. 4.3. Определение вектора половинного направления

Упражнения к главе 4

1. Найдите на поверхности сферы единичного радиуса четыре точки, находящиеся друг от друга на одинаковом расстоянии. *Указание.* Одну из точек можно выбрать произвольно; пусть это будет точка (0, 1, 0). В таком случае другие три лежат на плоскости $y = -d$, где *d* — некоторое положительное число.

2. Покажите, что если v лежит в той же плоскости, что и l, n и r, то угол половинного направления удовлетворяет условию $2\psi = \varphi$. Какое соотношение существует между этими углами в случае, если v не является компланарным остальным векторам?

3. Покажите, что вектор половинного направления h задает такую ориентацию поверхности, которая обеспечивает максимальное отражение светового потока к наблюдателю.

4. Найдите вектор преломленного луча, который определяет ориентацию поверхности, обеспечивающую максимальное отражение падающего луча в сторону наблюдателя.

5. Вычислите вектор отражения, если заданы вектор нормали и вектор падающего луча.

РАЗДЕЛ Б. ОСНОВЫ ПРОГРАММИРОВАНИЯ ГРАФИЧЕСКИХ ОБРАЗОВ

Глава 5. Графическая библиотека OpenGL

5.1. Подключение библиотеки OpenGL и начальная настройка формы проекта

Стандарт *OpenGL* был разработан как эффективный аппаратно независимый интерфейс, работающий на различных аппаратных платформах и под различными операционными системами. Поэтому OpenGL не включает в себя никаких специальных команд для работы с окнами или ввода информации от пользователя.

Библиотека OpenGL позволяет выполнять функции:

1) создавать объекты из геометрических примитивов (точки, линии, грани и битовые изображения);

2) располагать объекты в трехмерном пространстве и выбирать способ и параметры проецирования;

3) вычислять цвет всех объектов. Цвет может как задаваться, так и вычисляться с учетом источников света, параметров освещения, текстур;

4) переводить математическое описание объектов и связанной с ними информации о цвете в изображение на экране.

Стандарт реализации OpenGL для Windows требует выполнения некоторых настроек, связанных с особенностями операционной системы. Для того чтобы оконная система могла работать с OpenGL, необходимо провести ее инициализацию и сконфигурировать буфер фрейма.

Графическая программа, как и любое другое приложение Windows, нуждается в ссылке на окно HDC (контекст устройства), на которое будет осуществляться воспроизведение. Ссылка на контекст воспроизведения — величина типа HGLRC (Handle openGL Rendering Context) — связывает контекст воспроизведения OpenGL с контекстом устройства.

Таким образом, чтобы начать работать с командами OpenGL, приложение должно создать один или несколько контекстов воспроизведения для потока и сделать текущим один из них. Каждый поток при этом может иметь один и только один текущий контекст воспроизведения, который ассоциирован с определенным контекстом устройства.

Прежде чем получить контекст воспроизведения, драйвер OpenGL должен получить детальные характеристики используемого оборудования. Эти характеристики хранятся в специальной структуре, тип которой — *TPixelFormatDescriptor* (описание формата пикселей). Формат пикселей определяет число битов на пиксель, конфигурацию буфера цвета и вспомогательных буферов, используемых для вывода изображения.

5.2. Использование дополнительных библиотек

Несмотря на то что библиотека OpenGL предоставляет практически все возможности для моделирования и воспроизведения трехмерных сцен, некоторые из функций, необходимых при работе с графикой, напрямую отсутствуют в библиотеке. Например, чтобы задать положение и направление камеры, с которой будет наблюдаться сцена, нужно самому рассчитывать матрицу вида, а это далеко не простая задача. Поэтому для OpenGL существуют так называемые вспомогательные библиотеки.

Библиотека GLU уже стала стандартом и устанавливается вместе с основной библиотекой OpenGL. В состав этой библиотеки вошли более сложные функции, с помощью которых можно определить, например, цилиндр или диск. Также в библиотеку вошли функции для работы со сплайнами, реализованы дополнительные операции над матрицами и дополнительные виды проекций (задание перспективной проекции).

OpenGL Utility Toolkit (GLUT) — библиотека утилит для приложений под OpenGL, которая в основном отвечает за системный уровень операций ввода-вывода при работе с операционной системой. Это независимая от платформы библиотека. Она реализует не только дополнительные функции OpenGL, но и предоставляет функции для работы с окнами, клавиатурой и мышкой. С библиотекой GLUT всё построение графики намного упрощается; буквально несколькими командами можно определить окно, в котором будет работать OpenGL, определить прерывание от клавиатуры или мыши — и все это не будет зависеть от текущей операционной системы.

Библиотека *GLUT* предоставляет также функции, с помощью которых можно задать в пространстве сцены сложные правильные многогранники: куб, сферу, тор, конус, тетраэдр и додекаэдр — и даже можно с помощью одной команды `glutSolidTeapot (N)` определить сложный объект — чайник. Например, для воспроизведения куба достаточно выполнить команду `glutSolidCube (n)`, где *n* задает длину грани куба.

Для визуализации трехмерной сцены после загрузки окна формы (объект `SimpleOpenGLControl`) необходимо заполнить метод `Form1_Load`:

```
private void Form1_Load (object sender, EventArgs e)
```

После инициализации компонента вывода инициализируется библиотека GLUT. Выполняется это командой `glutInit ()`. Кроме того, необходимо задать режим работы компонента вывода средствами библиотеки. Для работы с функциями библиотеки GLUT используется класс `GLUT`.

```
//инициализация Glut
Glut.glutInit ();
Glut.glutInitDisplayMode (Glut.GLUT_RGB|Glut.GLUT_
DOUBLE|Glut.GLUT_DEPTH);
```

В параметрах инициализации библиотеки GLUT_RGB означает, что используется режим цветов RGB, GLUT_DOUBLE означает использование двойной буферизации и GLUT_DEPTH — использование буфера глубины (алгоритм Z-буфера).

После инициализации библиотеки GLUT необходимо установить цвет очистки окна, настроить освещение сцены, установить тестирование буфера глубины и обработки цвета при заданном освещении.

5.3. Синтаксис команд OpenGL

Все команды OpenGL начинаются с префикса *gl*, затем идет имя команды, цифра и суффикс. Цифра в окончании соответствует количеству аргументов, буква показывает требуемый тип аргумента.

Если имя команды заканчивается на *v* (векторная форма), то аргументом ее служит указатель на массив значений. Например, если последние три символа в имени команды *3fv,* то ее аргумент — адрес массива трех вещественных чисел.

В общем виде команду можно представить как

```
glCommanName{1,2,3,4}{b, s, i, f, d, ub, us, ui}{v}
(arguments)
```

В табл. 5.1 представлены возможные типы аргументов в функциях библиотеки.

Таблица 5.1

Возможные типы аргументов

Символ	Обозначение типа в OpenGL	Расшифровка
b	`GLbyte`	Байтовый
s	`GLshort`	Короткий целый
i	`GLint`	Целый
d	`GLdouble`	Вещественный двойной точности
f	`GLfloat`	Вещественный
ub	`GLubyte`	Байтовый беззнаковый
us	`GLushort`	Короткий целый беззнаковый
ui	`GLuint`	Целый беззнаковый

Почти всегда предпочтительно использовать команду в вещественной форме, поскольку хранит данные OpenGL именно в вещественном формате.

5.4. Рисование примитивов

Процедура рисования заключается в командные скобки:

```
glBegin (mode)
…//команды, указывающие вершины фигуры
glEnd ();
```

Главное назначение командных скобок — это задание режима, определяющего, как соединять точки (вершины). Вершины задаются своими координатами (количество координат зависит от пространства изображения) с помощью команды

glVertex{2,3,4}{s, i, f, d} (arg).

Режим (`mode`), задающий правило соединения точек, определяет примитив. К примитивам относятся точки, линии, связанные линии, замкнутые линии, треугольники, связанные треугольники, четырехугольники, связанные четырехугольники и многоугольники. Ниже представлены возможные режимы (mode) и приведено их описание:

`GL_POINTS` — каждый вызов `glVertex` задает отдельную точку. Выводится *N* точек;

`GL_LINES` — каждая пара вершин задает отрезок. Выводится *N*/2 отрезков;

`GL_LINE_STRIP` — выводится ломаная. Элементы *n* и *n*+1 определяют отрезок n. Выводится *N* — 1 отрезков;

`GL_LINE_LOOP` — рисуется ломаная, причем ее последняя точка соединяется с первой. Элементы *n* и *n* + 1 определяют отрезок *n*. Последняя линия определяется элементом *n* и 1. Выводится *N* отрезков;

`GL_TRIANGLES` — каждые три вызова `glVertex` задают треугольник. Элементы 3*n* — 2, 3*n* — 1, и 3*n* определяют треугольник *n*. Выводится *N*/3 треугольников;

`GL_TRIANGLE_STRIP` — рисуются треугольники с общей стороной. Для каждого *n* элементы *n*, *n* — 1, *n* — 2 определяют новый треугольник. Выводится *N* — 2 треугольника;

`GL_TRIANGLE_FAN` — рисует группу соединенных треугольников. Один треугольник определяется для каждого элемента после двух предыдущих. Два последних элемента соединяются с первым. Выводится *N* — 1 треугольник;

`GL_QUADS` — каждые четыре вызова `glVertex` задают четырехугольник. Выводится *N*/4 четырехугольника;

`GL_QUAD_STRIP` — выводятся четырехугольники с общей стороной. Каждый четырехугольник треангулируется: для каждого *n* элементы *n*, *n* — 1, *n* — 2 определяют новый треугольник;

`GL_POLYGON` — элементы с 1 по *N* определяют полигон (при этом точки полигона сортируются так, чтобы грани у получившегося многоугольника не пересекались).

Работая с библиотекой OpenGL, можно связать нормаль с некоторой вершиной с помощью функций:

```
glNormal3f (nx, ny, nz); glNomal3fv (pointer_to_
normal);
```

Аргументы функции `glNormal3f ()` — компоненты вычисленного раньше вектора нормали, а функция `glNormal3fv ()` использует в качестве аргумента указатель на массив компонентов уже сформированного вектора. Нормали в OpenGL являются переменными. Если определить нормаль перед тем, как задавать новые вершины с помощью `glVertex ()`, то эта нормаль будет связана со всеми вершинами, сформированными далее по ходу выполнения программы. Однако проблему вычисления нормали должен решать прикладной программист.

5.5. Визуализация сцены

Процесс создания сцены начинается с позиционирования области видимости в пространстве окна.

```
Gl.glViewport (0, 0, AnT. Width, AnT. Height);
```

Следующим шагом создания сцены будет задание типа проецирования и области вывода, которая оказывается в поле зрения наблюдателя. Заключительным шагом является построение объектов в рамках сцены.

Если область вывода не задана явно, то в OpenGL используется установленная по умолчанию зона в виде куба видимости 2×2×2 с началом координат в центре куба (рис. 5.1).

Система координат *Oxyz* на рисунке расположена таким образом, что ось *Oz* направлена в сторону, противоположную направления зрения. Окно видимости (*Windows*) масштабируется в пределах [–1; 1] по осям *Ox*, *Oy*. Изображение по умолчанию воспроизводится на плоскости $z = 0$.

В графической системе существует два типа проецирования: параллельная проекция и перспективная. Ортогональная проекция — это частный случай параллельной проекции, при которой проецирующие лучи ортогональны картинной плоскости.

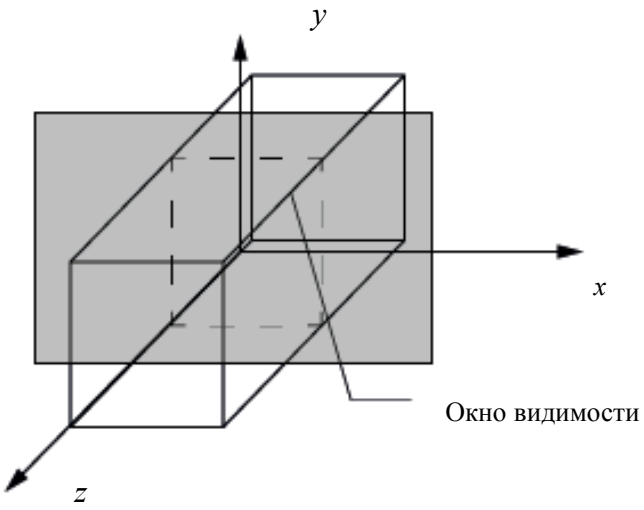

Рис. 5.1. Вид системы координат Oxyz

При ортогональном проецировании точка объекта (*x, y, z*) проецируется в точку (*x, y*, 0) на плоскости проекции. В OpenGL ортогональная проекция, характеризуемая параллелепипедом видимости, задается функцией `GL.glOrtho ()`, объявленной следующим образом:

`Gl.glOrtho (left, right, bottom, top, near, far)` — видны будут объекты, которые попали внутрь параллелепипеда видимости (рис. 5.2).

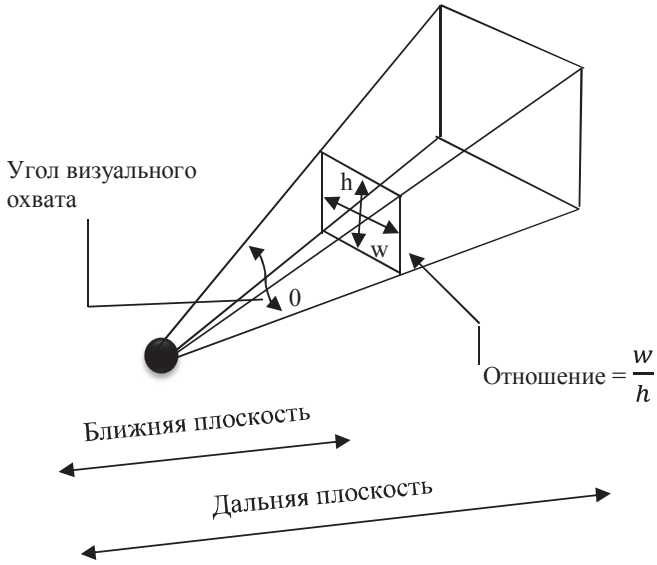

Рис. 5.2. Параллелепипед видимости

Перспективное проецирование устанавливается с помощью функции `gluPerspective ()`. Данная функция строит пирамиду видимости (рис. 5.3), основываясь на угле визуального охвата, отношении сторон картинной плоскости и установке ближней и дальней плоскости отсечения.

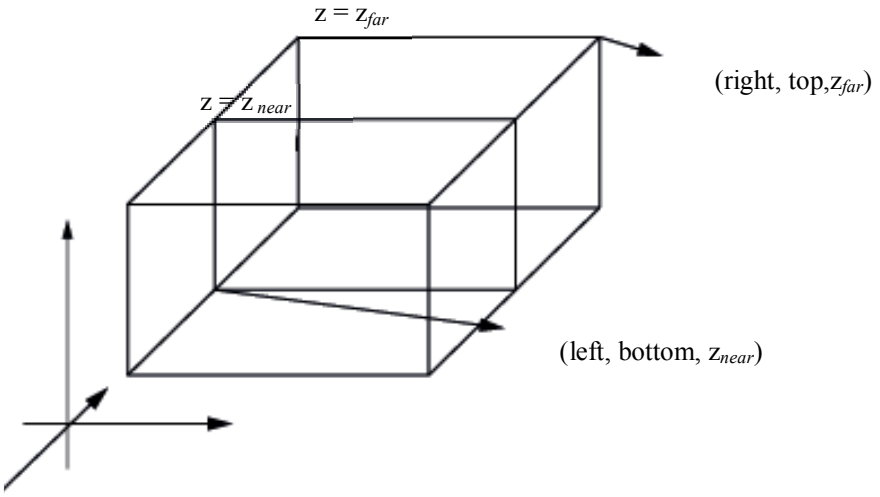

Рис. 5.3. Пирамида видимости

```
Glu.gluPerspective (fovy, Width/Height, near, far);
```

где `fovy` — угол обзора по оси *y*; `Width/Height` — отношение ширины окна картинной плоскости к его высоте; `near` — расстояние от центра проецирования до ближней отсекающей плоскости; `far` — расстояние от центра проецирования до задней плоскости отсечения.

В ходе проецирования преобразование координат объектов включает в себя этапы, изображенные на рис. 5.4. Сначала мировые координаты (система координат, в которой определяется положение объекта, положение точки наблюдения и экрана) преобразовываются в видовые координаты. При этом точки изображения остаются на своих местах, но система мировых координат переходит в систему видовых координат. Затем выполняется проективное преобразование, добавляющее эффект перспективы (искажения) в зависимости от расстояния от объекта до экрана и расстояние от точки наблюдения до экрана. Система трехмерных видовых координат переходит в систему нормализованных координат.

Рис. 5.4. Схема преобразования координат

Все преобразования объектов и камеры в *OpenGL* производятся с помощью умножения векторов координат на матрицы, причем умножение происходит на текущую матрицу.

Область вывода представляет собой прямоугольник в оконной системе координат, размеры которого задаются командой:

```
Gl.glViewPort (x, y, width, height);
```

Значения всех параметров этой функции задаются в пикселях. Они определяют ширину и высоту области вывода с координатами левого нижнего угла (*x*, *y*) и правого верхнего угла (*width, height*) в оконной системе координат. Используя параметры команды `glViewPort()`, OpenGL вычисляет оконные координаты каждой вершины (x_n, y_n) по формуле

$$\begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} \left(\frac{width}{2}\right) x_n + o_x \\ \left(\frac{height}{2}\right) y_n + o_y \\ \left(\frac{far - near}{2}\right) z_n + \left(\frac{near + far}{2}\right) \end{bmatrix}.$$

где (o_x, o_y) — оконные координаты центра области вывода. При этом целые положительные величины *near* и *far* задают минимальную и максимальную глубину точки в окне. Глубина каждой точки записывается в специальный буфер глубины (*z*-буфер).

5.6. Видовые преобразования

Видовые преобразования осуществляются с помощью матричных преобразований. В библиотеке OpenGL текущая матрица преобразований является произведением двух матриц — матрицы модели и матрицы проецирования, при этом формируется единая матрица преобразования, которая применяется ко всем вершинам всех геометрических объектов.

Функция `glMatrixMode ()` предназначена для того, чтобы задавать матричный режим.

Матрица модели — `Gl.glMatrixMode (GL.GL_MODELVIEW)` — связана с координатами объектов. Она задается в базисе видовых (трехмерных) координат и используется для построения изображения в том виде, как ее представляет наблюдатель.

Матрица проецирования — `GL.glMatrixMode (GL.GL_PROJECTION)`. Эта матрица в системе координат устройства вывода. В ней хранятся нормализованные координаты, которые преобразуются в экранные после трансформаций, связанных с областью вывода.

Функция `GL.glLoadIdentity ()` заменяет текущую матрицу единичной.

Фрагмент настройки проекции:

```
  Gl.glMatrixMode (Gl.GL_PROJECTION);
  Gl.glLoadIdentity ();
  Glu.gluPerspective (45,
(float)pir.Width/(float)pir.Height, 0.1, 200);
  Gl.glMatrixMode (Gl.GL_MODELVIEW);
  Gl.glLoadIdentity ();
```

Для корректной визуализации сцены необходимо только включить некоторые опции. Это — тест глубины, а также отображение цветов:

```
Gl.glEnable (Gl.GL_DEPTH_TEST); Gl.glEnable (Gl.GL_
COLOR_MATERIAL);
```

Функция визуализации вызывается каждый раз при отрисовке кадра динамической сцены средствами какой-либо библиотеки или среды разработки. Она состоит:

- из описания параметров вывода через визуальный компонент;
- описания модели;
- описания текстурирования;
- команд завершения визуализации.

Работу с освещением добавляют с помощью функций `Gl.glEnable (Gl.GL_LIGHTING)` и «включают» нулевой источник света — `Gl.glEnable (Gl.GL_LIGHT0)`.

Закраска объектов сцены производится с помощью задания функций материала:

```
Gl.glMaterialfv (Gl.GL_FRONT, Gl.GL_DIFFUSE,
color);//цвет диффузного отражения.
Gl.glMaterialfv (Gl.GL_FRONT, Gl.GL_SPECULAR,
color);//цвет зеркального отражения.
Gl.glMaterialfv (Gl.GL_FRONT, Gl.GL_SHININESS,
shininess);//степень зеркального отражения.
Gl.glMaterialfv (Gl.GL_FRONT, Gl.GL_AMBIENT,
color);//цвет фонового освещения.
```

5.7. Аффинные преобразования

Масштабирование

Преобразование масштабирования увеличивает или уменьшает размеры объекта.

Функция масштабирования `GL.glScale (arg1, arg2, arg3)` имеет три аргумента — коэффициенты масштабирования по каждой из осей. Если масштабные множители больше единицы, объект растягивается в заданном направлении, если меньше — объект сжимается. Масштабные множители могут иметь отрицательные значения, при этом изображение переворачивается по соответствующей оси. При двумерных построениях значение коэффициента по оси *z* игнорируется.

После любой функции масштабирования следует восстановить нормальный масштаб, чтобы каждое следующее обращение к обработчи-

ку перерисовки экрана не приводило бы к последовательному уменьшению или увеличению изображения.

Поворот

Для поворота изображения используется функция `glRotate (arg1, arg2, arg3, arg4)` с четырьмя аргументами: `arg1` задает угол поворота в градусах и `arg2, arg3, arg4` задают вектор поворота.

Сдвиг

Преобразование сдвига смещает точки в новые позиции в соответствии с заданным вектором смещения. Перенос системы координат осуществляется функцией `glTranslate (arg1, arg2, arg3);` `arg1, arg2, arg3` — величины переноса по каждой из осей.

Для поворота вокруг произвольной фиксированной точки сначала нужно выполнить преобразование сдвига, совмещающего заданную фиксированную точку с началом координат фрейма, потом выполнить преобразование поворота вокруг начала координат, а затем обратное преобразование сдвига. Порядок манипуляции с системой координат следующий: перенос, затем поворот; после отрисовки объектов сцены — в обратном порядке: поворот, затем перенос.

Глава 6. Лабораторный практикум

Для успешного выполнения лабораторного практикума необходимо проявить максимум творчества и фантазии.

В результате выполнения семи лабораторных работ должен получиться анимационный трехмерный мультфильм. Конкретные элементы сцены, их расположение, взаимодействие и параметры определяются разработчиком исходя из его цели и эстетических соображений.

Правила оформления отчета

Отчет по лабораторным работам должен содержать следующие структурные части:

- титульный лист;
- содержание;

- цель и задачи лабораторной работы;
- формулировку задания;
- ход работы;
- программную реализацию;
- результаты работы;
- заключение;
- приложение (текст основного модуля программы с комментариями);
- библиографический список.

Титульный лист содержит основные сведения о работе, название университета, кафедры, фамилию автора работы, фамилию преподавателя, дату и год выполнения работы.

Содержание отражает структурные части отчета.

Цели и задачи лабораторной работы формулируются в соответствии с заданием и методическими указаниями для выполнения работы.

Формулировка задания содержит условия индивидуального варианта задания, раскрывает постановку задачи.

Ход работы описывает пошаговое выполнение лабораторной работы: выбор среды разработки приложения, языка программирования, перечень используемых в программе библиотек и функций.

Раздел «Программная реализация» содержит описание алгоритмических фрагментов кода программы, основные решения визуализации графических образов.

Результаты работы оформляются в виде снимков экрана, демонстрирующих работу графического приложения.

В заключении необходимо указать достижение цели и выполнение задач лабораторной работы.

Приложения оформляются как структурные части отчета. Каждое приложение начинается с нового листа. При ссылке на приложение пишут слово «приложение» полностью и указывают номер приложения, например: «... в приложении 3». Единственное в отчете приложение нумеруется.

В соответствии с ГОСТ 7.1–2003 оформляется библиографический список.

Нумерация всех листов отчета с приложениями должна быть сквозная.

Лабораторная работа № 1. Синусоида

Цель работы. Научиться создавать проекты, использующие графическую библиотеку OpenGL, изучить различные режимы отрисовки и закраски примитивов.

Задание. Создать проект, использующий библиотеку OpenGL. Написать процедуру рисования синусоиды в координатных осях.

Методические рекомендации

Любой проект начинается с подключения библиотек:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using Tao.OpenGl;
using Tao.Platform.Windows;
```

Для работы с функциями библиотеки OpenGL используется класс Gl, находящийся в пространстве имен `Tao.OpenGL`.

Для отображения сцены необходимо разместить элемент, с помощью которого библиотека OpenGL выводит изображение на рабочее пространство приложения (в данном случае — форму). Для этого в библиотеке `Tao.Platform.Windows` существует визуальный компонент `simpleOpenGLControl`, который необходимо разместить на форме.

Команда <имя компонента> `InitializeComponent ()` используется для инициализации компонента отображения `simpleOpenGLControl`. Данная команда располагается в конструкторе класса формы и инициализирует работу компонента непосредственно после запуска приложения.

```
public TaoName ()
        {
             InitializeComponent ();
             name.InitializeContexts ();}
```

Для трехмерного случая необходимо задать область вывода и способ проецирования.

Лабораторная работа № 2. Куб

Цель работы. Научиться создавать трехмерную сцену, применять различные аффинные преобразования к объектам сцены.

Задание. Создать трехмерную сцену. Задать область вывода объектов и способ проецирования. Создать в пространстве графический объект — куб. Для каждой грани куба указать вектор нормали, определяющий лицевую поверхность. Применить аффинные преобразования: поворот, перенос, отражение.

Методические рекомендации

Для задания способа проецирования используйте функции:

```
glOrtho (left, right, bottom, top, near, far);
gluPerspective (fovy, aspect, near, far);
glFrustum (xmin, xmax, ymin, ymax, near, far).
```

Значения аргументов `near, far` должны быть положительными и отсчитываться от центра проецирования вдоль оси проецирования.

Аффинные преобразования выполняются с помощью функций

```
glScalef (arg1, arg2, arg3);
glRotatef (arg1, arg2, arg3, arg4);
glTranslatef (arg1, arg2, arg3).
```

Для закрашивания объектов сцены процедуру рисования (до вывода примитивов) необходимо дополнить функцией

```
glMaterialfv (GL_FRONT,
   GL_AMBIENT_AND_DIFFUSE, MaterialColor);
```

Фронтальная плоскость имеет характеристики рассеянного и диффузного света, заданные массивом RGBA в константе `MaterialColor`.

```
GLfloat MaterialColor [4] = {0.5, 0.6, 1.0, 1.0};
```

Лабораторная работа № 3. Анимация

Цель работы. Научиться создавать трехмерные объекты, состоящие из разных примитивов, и использовать различные способы закраски и отображения полигонов. Научиться моделировать тень от объекта на горизонтальную поверхность, использую матрицу проекции на плоскость.

Задание. Создать трехмерный графический объект, применить к нему анимацию с использованием графического стандарта OpenGL и библиотек GLU и GLUT. Для объектов сцены создать тень на горизонтальной поверхности.

Методические рекомендации

Примеры использования библиотеки GLUT:

```
glutSolidCube (2);//куб с ребром 2.
glutSolidSphere (1.5, 20, 20);//сфера с параметрами (радиус, количество меридиан, количество сегментов).
glutSolidTeapot (1.5);//чайник (1,5 — радиус описанной окружности).
```

Пример функции для отрисовки цилиндра с помощью библиотеки GLU:

```
void DrawCylinder (GLDouble radius, height)
{
GLUquadricObj quadr;
   quadr=gluNewQuadric ();
   gluCylinder (quadr, radius, radius, height,10,2);
}
```

Способ отображения полигонов задается функцией

```
glPolygonMode (GLenum face, Glenum mode)
```

Параметр *mode* определяет, как будут отображаться многоугольники, а параметр *face* устанавливает тип многоугольников, к которым будет применяться эта команда. Значения параметров представлены в табл. 6.1.

Таблица 6.1

Значения параметров отображения полигонов

Параметры отображения		Значения параметров
GLenum face	GL_FRONT	Для лицевых граней
	GL_BACK	Для обратных граней
	GL_FRONT_AND_BACK	Для всех граней
Glenum mode	GL_POINT	Отображаются вершины многоугольников
	GL_LINE	Представляется набором отрезков
	GL_FILL	Закрашиваются текущим цветом с учетом освещения

Свойства материала задаются для внешней и внутренней стороны полигона:

```
glMaterialfv (GL_FRONT, GL_AMBIENT_AND_DIFFUSE, param);
```

```
glMaterialfv (GL_BACK, GL_AMBIENT_AND_DIFFUSE,
param);
```

С помощью указанных функций можно определить рассеянную, диффузную и зеркальную составляющие материала, а также степень зеркального отражения и интенсивность излучения света, если поверхность объекта светится. Ниже описаны возможные характеристики взаимодействия света с материалом:

`GL_AMBIENT` — рассеянный свет, параметр задает четыре значения цветов (RGBA). Значение по умолчанию (0.2, 0.2, 0.2, 1.0);

`GL_DIFFUSE` — отраженный свет, параметр задает четыре значения цветов (RGBA). Значение по умолчанию (0.8, 0.8, 0.8, 1.0);

`GL_EMISSION` — излучаемый свет, параметр задает четыре значения цветов (RGBA). Значение по умолчанию (0.0, 0.0, 0.0, 1.0);

`GL_SPECULAR` — зеркальный блик, параметр задает четыре значения цветов (RGBA). Значение по умолчанию (0.0, 0.0, 0.0, 1.0);

`GL_SHININESS` — степень зеркального отражения материала, задается значением в диапазоне от 0 до 128. Значение по умолчанию 0.

Для отрисовки проекции объекта на плоскость необходимо задать матрицу проецирования. Произвольную матрицу размером 4×4 можно задать как одномерный массив из 16 чисел, в котором элементы матрицы перечислены по столбцам. Текущую матрицу вида необходимо умножить справа на заданную с помощью функции умножения `glMultmatrixf (<указатель на матрицу>)`. Если необходимо нарисовать проекцию объекта на плоскость, нужно задать матрицу плоскости и выполнить умножение на нее текущей матрицы вида.

После этой функции все объекты будут выводиться в преобразованной матрице. Чтобы отключить режим проекции, необходимо выполнить обратное преобразование или загрузить в матрицу модели единичную матрицу.

Лабораторная работа № 4. Свет

Цель работы. Научиться создавать в графическом приложении источники света с различными характеристиками.

Задание. Расположить в произвольных точках пространства несколько источников света. Задать характеристики источников света: один источник должен быть точечный, другой — направленный (типа «прожектор»). Цвет источников подбирается из эстетических соображений.

Методические рекомендации

Для того чтобы инициализировать источник и включить обработчик расчета воздействия источника на объекты, достаточно выполнить команды

```
glEnable (gl_lighting);
glEnable (gl_light0);
```

Источник света по умолчанию является удаленным и располагается в пространстве с координатами (0, 0, ∞). Можно создавать источник света в любой точке пространства сцены. Для этого ему его необходимо задать соответствующие параметры. Параметры источника света задаются с помощью функции

```
glLightfv (source, parameter, pointer_to_array).
```

Первый параметр команды — идентификатор источника. Идентификатор задается с помощью символьной константы, например, `GL_LIGHT1`. Второй аргумент — символьная константа, задающая атрибут (характеристику источника света). Третий — ссылка на структуру, содержащую задаваемые значения для данного атрибута. Далее представлены характеристики функции цвета:

`GL_Position` — задает позицию источника света, (*x, y, z, w*). Если значение компоненты *w* равно 0.0, то источник считается бесконечно удаленным, и при расчете освещенности учитывается только направление на точку (*x, y, z*), и ослабление света при удалении от источника не происходит. Значение по умолчанию (0.0, 0.0, 1.0, 0.0);

`GL_AMBIENT` — задает цветовое значение, соответствующее рассеянному свету (RGBA). Значение по умолчанию (0.0, 0.0, 0.0, 1.0);

`GL_DIFFUSE` — задает цветовое значение, соответствующее диффузному свету (RGBA). Значение по умолчанию (1.0, 1.0, 1.0, 1.0) для GL_LIGHT0 и (0.0, 0.0, 0.0, 1.0) для остальных источников света;

`GL_SPECULAR` — задает цветовое значение, соответствующее зеркальному блику. Значение по умолчанию (1.0, 1.0, 1.0, 1.0) для GL_LIGHT0 и (0.0, 0.0, 0.0, 1.0) для остальных;

`GL_SPOT_DIRECTION` — определяет вектор направления света (*x, y, z*). Характеристика источника имеет смысл, если значение GL_SPOT_CUTOFF отлично от 180;

`GL_SPOT_CUTOFF` — максимальный угол излучения направленного источника света, передает угол в градусах в диапазоне [0,90] или 180°. Значение этого параметра есть половина угла в вершине конусовид-

ного светового потока, создаваемого источником. Значение по умолчанию 180 (рассеянный свет);

GL_SPOT_EXPONENT — уровень сфокусированности источника света, целое или вещественное число от 0 до 128, задающее распределение интенсивности света. Значение по умолчанию 0 (рассеянный свет);

GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, GL_QUADRATIC_ATTENUATION — задает значения коэффициентов, определяющих ослабление интенсивности света при удалении от источника. Допускаются только неотрицательные значения. По умолчанию эти параметры задаются (1, 0, 0), и фактически ослабление не происходит.

При изменении положения источника света следует учитывать, что в OpenGL источники света являются объектами во многом такими же, как многоугольники и точки. На них распространяется основное правило обработки координат: параметры, задающие положение в пространстве, преобразуются текущей модельно-видовой матрицей в момент формирования объекта, т. е. в момент вызова соответствующих команд OpenGL. Таким образом, формируя источник света одновременно с объектом сцены или камерой, его можно привязать к этому объекту или, наоборот, сформировать стационарный источник света, который будет оставаться на месте, пока другие объекты перемещаются.

Для задания глобальных параметров освещения используются функции

```
glLightModel [i f] (pname, param);
glLightModel [i f]v (pname, const GLtype *params);
```

Аргумент pname определяет параметр модели освещения и может принимать следующие значения:

GL_LIGHT_MODEL_LOCAL_VIEWER — параметр param должен быть булевым, он задает положение наблюдателя. Если параметр равен GL_FALSE, то направление обзора считается параллельным оси $-z$ вне зависимости от положения в видовых координатах. Если он равен GL_TRUE, то наблюдатель находится в начале видовой системы координат. Это может улучшить качество освещения, но усложняет его расчет. Значение по умолчанию GL_FALSE.

GL_LIGHT_MODEL_TWO_SIDE — параметр param должен быть булевым, он управляет режимом расчета освещенности для внутренней стороны объекта. Если он равен GL_FALSE, то освещенность рассчитывается только для лицевых граней. Если же параметр равен GL_

`TRUE`, расчет проводится и для внутренних поверхностей. Значение по умолчанию `GL_FALSE`.

`GL_LIGHT_MODEL_AMBIENT`— параметр `params` должен содержать четыре целых или вещественных числа, которые определяют цвет фонового освещения даже в случае отсутствия определенных источников света. Значение по умолчанию (0.2, 0.2, 0.2, 1.0).

Лабораторная работа № 5. Прозрачность

Цель. Научиться создавать прозрачные объекты, используя α-канал и смешивание цветов.

Задание. Включите в ранее созданный проект полупрозрачный объект. Отрисовка этого объекта должна быть самой последней в процедуре рисования сцены.

Методические рекомендации

Для того чтобы создавать прозрачные и полупрозрачные объекты, необходимо разрешить тестировать буфер альфа-канала и включить механизм альфа-смешивания.

Интенсивность альфа-канала входит в качестве четвертой компоненты в вектор цвета в формате RGBA. Коэффициент поглощения α определенной среды есть количественная мера, позволяющая оценить, какая доля светового потока, падающего на поверхность раздела сред, проходит далее, а какая — поглощается: $A = 1$ — свет полностью поглощается; $A = 0$ — весь падающий поток проходит дальше (полностью прозрачная поверхность).

Коэффициент прозрачности (*transparency*) поверхности связан с коэффициентом поглощения по зависимости $\tau = 1 - \alpha$.

Для OpenGL заданное значение интенсивности компоненты *A* определяется коэффициентом перерасчета закраски пикселя при записи его в буфер кадра. В результате оказывается, что в код закраски пикселя вносит свой вклад несколько геометрических объектов, и в итоге формируется смешанное изображение. Использование альфа-канала является одним из способов выполнить смешение цветов на уровне отдельных пикселей.

Настройка режима смешения изображений в OpenGL выполняется с помощью функции `glEnable (GL_BLEND);` затем выполняется настройка значений коэффициентов смешения.

```
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

При включенном альфа-тесте сравнивается входящее значение альфа-канала с эталонным значением. Фрагмент принимается или откло-

няется в зависимости от результатов сравнения. Эталонное значение и функция сравнения задаются командой `glAlphaFunc ()`. По умолчанию эталонное значение равно 0, установлена функция сравнения `GL_ALWAYS` и альфа-тест отключен.

Для расчета прозрачности процедуру отрисовки дополните строчками:

```
glEnable (GL_ALPHA_TEST);
glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

Основная сложность в управлении процессом формирования смешанного изображения состоит в том, что на результат влияет порядок построения объектов 3D-сцены. Коэффициент смешивания пикселя источника приравнивается к значению α, а коэффициент смешивания приемника — к значению $(1 - \alpha)$. В результате получим

$$(R_d, G_d, B_d, A_d) = (\alpha_s R_s + (1-\alpha_s)R_d, \alpha_s G_s + (1-\alpha_s)G_d, \alpha_s B_s + (1-\alpha_s)B_d, \alpha_s\alpha_d + (1-\alpha_s)\alpha_d).$$

Альфа-тест отбрасывает фрагменты в зависимости от результата сравнения между поступившим значением альфа и некоторой указанной константой. Определить константу и функцию сравнения можно с помощью команды `glAlphaFunc`. Сравнение производится только в том случае, если разрешено альфа-тестирование.

Функция сравнения альфа-компоненты

```
glAlphaFunc (func, ref);
```

Параметры `func` и `ref` определяют условия, по которым отображается пиксель. Поступившее значение альфа сравнивается с константой `ref` при помощи функции `func`. Если сравнение проходит, входящий фрагмент отображается при условии последующего тестирования по буферу трафарета. Если сравнение не проходит, Z-буфер с координатами этого пикселя не изменяется. Ниже приведено описание функции `func`*:*

`GL_NEVER` — никогда не обрабатывать;

`GL_LESS` — обрабатывать, если входящее значение альфа меньше указанного;

`GL_EQUAL` — обрабатывать, если входящее значение альфа равно указанному;

`GL_LEQUAL` — обрабатывать, если входящее значение альфа меньше или равно указанного;

`GL_GREATER` — обрабатывать, если входящее значение альфа больше указанного;

GL_NOTEQUAL — обрабатывать, если входящее значение альфа не равно указанному;

GL_GEQUAL — обрабатывать, если входящее значение альфа больше или равно указанного;

GL_ALWAYS — всегда обрабатывать. Установлено по умолчанию.

Значение, с которым сравнивается альфа-компонент, — *ref*. Это значение ограничено диапазоном от 0 до 1, где 0 представляет минимальное возможное значение альфа и 1 — максимально возможное значение. По умолчанию применяется значение 0.

Функция glAlphaFunc обрабатывает запись всех пикселей, включая преобразования развертки точек, линий, полигонов, растров. Функция glAlphaFunc не влияет на операцию очистки экрана.

Если в сцене есть несколько прозрачных объектов, которые могут перекрывать друг друга, корректный вывод можно гарантировать только в случае выполнения следующих условий:

- все прозрачные объекты выводятся после непрозрачных;
- при выводе объекты с прозрачностью должны быть упорядочены по уменьшению глубины.

Функция glBlendFunc определяет пиксельную арифметику

glBlendFunc (Sfactor, dfactor);

Sfactor определяет, как вычисляются факторы смешивания исходных данных красного, зеленого, синего и альфа-каналов. Применяются девять символьных констант:

GL_ZERO, GL_ONE, GL_DST_COLOR, GL_ONE_MINUS_DST_COLOR, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA, GL_ONE_MINUS_DST_ALPHA и GL_SRC_ALPHA_SATURATE.

Dfactor определяет как вычисляются факторы смешивания назначенного значения для красного, зеленого, синего и альфа-каналов. Применяются восемь символьных констант:

GL_ZERO, GL_ONE, GL_SRC_COLOR, GL_ONE_MINUS_SRC_COLOR, GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA, GL_DST_ALPHA и GL_ONE_MINUS_DST_ALPHA.

В режиме RGB пиксели могут отображаться при помощи функции, которая смешивает входящее значение RGBA со значением RGBA, уже находящимся во фрейм-буфере. По умолчанию смешивание запрещено. Для разрешения и запрета смешивания используются функции glEnable и glDisable с аргументом GL_BLEND. Когда разрешено, glBlendFunc определяет операцию смешивания. Параметр sfactor

указывает, какой из девяти методов используется при обработке исходных компонентов цвета. Параметр `dfactor` указывает, какой из восьми методов используется при обработке назначенного цвета.

Ниже описаны одиннадцать возможных методов. Каждый метод определяет четыре масштабных коэффициента, по одному для красного, зеленого, синего и альфа-компонента. Исходный и назначенный цвет обозначены (R_S, G_S, B_S, A_S) и (R_d, G_d, B_d, A_d) соответственно. Предполагается, что они имеют целые значение между 0 и (k_R, k_G, k_R, k_A),

где $k_R = 2^{mR} - 1$;

$k_G = 2^{mG} - 1$;

$k_B = 2^{mB} - 1$;

$k_A = 2^{mA} - 1$.

Здесь (mR, mG, mB, mA) — количество битовых плоскостей для красного, зеленого, синего и альфа-компонентов.

Символьная константа методов смешивания цветов и их значения ($f(R), f(G), f(B), f(A)$):

```
  GL_ZERO..................(0,0,0,0)
  GL_ONE...................(1,1,1,1)
  GL_SRC_COLOR.............(RS/kR,  GS/kG,  BS/kB,
AS/kA)
  GL_ONE_MINUS_SRC_COLOR...(1,1,1,1) – (RS/kR, GS/kG,
BS/kB, AS/kA)
  GL_DST_COLOR.............(Rd/kR,  Gd/kG,  Bd/kB,
Ad/kA)
  GL_ONE_MINUS_DST_COLOR...1,1,1,1)
  GL_SRC_ALPHA.............AS/kA, AS/kA, AS/kA, AS/kA
  GL_ONE_MINUS_SRC_ALPHA...(1,1,1,1) – (AS/kA, AS/kA,
AS/kA, AS/kA)
  GL_DST_ALPHA.............(AD/kA,  AD/kA,  AD/kA,
AD/kA)
  GL_ONE_MINUS_DST_ALPHA...(1,1,1,1) – (AD/kA, AD/kA,
AD/kA, AD/kA)
  GL_SRC_ALPHA_SATURATE....(i, i, i,1)
```

Здесь $i = \min(A_S, k_A - A_D)/k_A$

Прозрачность лучше всего реализуется вызовом функции `glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)` с примитивами, отсортированными от дальнего к ближнему.

Лабораторная работа № 6. Текстуры

Цель. Изучить возможности библиотеки OpenGL для моделирования реалистичных изображений.

Задача. Написать приложение, в котором формируются одномерная и двухмерная текстуры, наложенные на объекты сцены. Один из объектов должен быть замкнутой фигурой, другой — поверхностью.

Методические рекомендации

Для решения поставленной задачи необходимо выполнить следующие действия.

Изучить функции библиотеки OpenGL: `glTexlmage1D`, `glTexlmage2D` и `glTexParameter`.

В ранее созданном приложении сформировать не менее двух текстур. Привязать текстуры к объектам сцены, используя функцию `glTexCoord2d ()` с соответствующими координатами текстуры. Изменяя параметры текстуры, проанализировать их влияние на визуализацию объектов и добиться оптимального результата.

Для того чтобы создавать двухмерную текстуру, необходимо сформировать ее в памяти компьютера.

В OpenGL предусмотрены две команды создания текстуры: одна для одномерного и вторая для двумерного вариантов образа.

```
glTexlmage1D (                  glTexlmage2D (
GLenum target,                  GLenum target,
GLint level,                    GLint level,
GLint components,               GLint components,
GLsizei width,                  GLsizei width,
GLint border,                   GLsizei height,
GLenum format,                  GLint border,
GLenum type,                    GLenum format,
const GLvoid* pixels)           GLenum type,
                                const GLvoid* pixels)
```

Параметры команд:

target определяет тип создаваемой текстуры и должен быть равен GL_TEXTURE_1D или GK_TEXTURE_2D соответственно;

level определяет число уровней детализации текстуры — 0 — базовый уровень, а *k* — уменьшенный в *k* раз образ;

components задает число цветовых компонентов текстуры и может принимать значения 1, 2, 3 или 4. При значении 1 используется толь-

ко красный компонент, при 2 — красный и альфа, при 3 — красный, зеленый и синий и при 4 — все четыре компонента цвета;

width определяет ширину образа текстуры и должен составлять 2^m+ 2 * *border* (на бордюр);

height определяет высоту образа текстуры, должен составлять 2^m + 2 * *border* (на бордюр) для команды *glTexImage2D* и всегда равен 1 для команды *glTexImage 1D;*

border определяет ширину границы и должен иметь значение 0 или 1.

format определяет формат данных пикселя и может принимать одно из следующих значений:

GL COLOR INDEX — каждый элемент представляет индекс цвета;

GL_RED — каждый элемент представляет собой единственный компонент — красный, зеленый и синий компоненты равны 0.0, а альфа — 1.0;

GLGREEN — только для зеленого цвета. Красный и синий компоненты равны 0.0;

GL_BLUE — только для синего цвета. Красный и зеленый компоненты равны 0.0;

GL ALPHA — только для компонента альфа. Остальным компонентам цвета присваиваются значения 0.0;

GL_RGB — каждый элемент представляет собой три компонента — красный, зеленый и синий, который преобразуется к вещественному формату, после чего собирается RGB, компонент альфа равен 1.0;

GL_RGBA — аналогично GL_RGB, только для всех четырех компонентов цвета;

GL_LUMINANCE — каждый элемент представляет собой единственный компонент — яркость, который преобразуется к вещественному формату, после чего собирается элемент RGBA, у которого компонент альфа равен 1.0;

GL_LUMINANCE_ALPHA — аналогично GLJ-UMINANCE, только явно задаются все четыре компонента цвета;

type определяет тип данных пикселя и может принимать одно из следующих значений — GL_UNSIGNED_BYTE, GL_BYTE, GL_UNSIGNED_INT, GL_INT, GL_UNSIGNED_SHORT, GL_SHORT, GL_BITMAP и GL_FLOAT;

pixels определяет указатель на образ данных в памяти.

При создании текстуры можно определить несколько образов с различным разрешением. Если текстура имеет размер $2^n \times 2^m$, то можно

определить $\max\{n, m\} + 1$ уменьшенных массивов. Первый имеет размер $2^n \times 2^m$, второй — $2^{n-1} \times 2^{m-1}$ и т. д., пока последний не будет иметь размер 1×1. Команды *glTexImage*D* предоставляют возможность определить $p = \max\{n, m\}$ таких массивов, в каждом из которых хранится уменьшенный образ исходного изображения. Наличие таких массивов позволит OpenGL использовать меньший образ для меньшего объекта, а больший — для большего.

Образ создается в пространстве текстуры с координатной системой (s, t), причем значения s и t находятся в отрезке [0,1] (рис. 6.1).

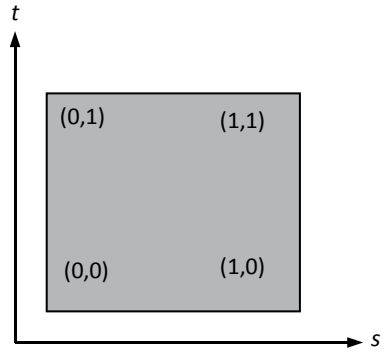

Рис. 6.1. Текстурные координаты

Один элемент на экране может покрывать несколько элементов массива образа, и, чтобы избежать проблем, связанных с лестничным эффектом, необходимо учитывать все затрагивающие этот массив элементы. Для этого определяются четыре точки в массиве образа, которые отображаются в четыре угла элемента на экране. Такие точки соединяются, и образуется четырехугольник. Значения попадающих в него элементов взвешиваются с учетом доли каждого элемента, содержащейся в многоугольнике, и затем суммируются.

Для учета особенностей текстуры необходимо установить параметры текстуры с помощью команды

`glTexParameter [i, f, v] (target, pname, param)`,

в которой `target` определяет, с какой текстурой предполагается работать — одномерной или двухмерной (GL_TEXTURE_1D или GL_TEXTURE_2D); `pname` определяет символическое имя параметра текстуры.

GL TEXTURE_MIN FILTER определяет функцию уменьшения текстуры, используется, когда площадь пикселя, на который она накла-

дывается, больше, чем элемент текстуры. Всего определено шесть таких функций. Две из них, определенные константами (GL_NEAREST и GL_LINEAR), используют один или четыре ближайшие элемента текстуры для расчета значения текстуры. Остальные четыре — уровни детализации.

Функция увеличения текстуры GL TEXTURE MAG FILTER используется, когда площадь пикселя, на который она накладывается, меньше или равна элементу текстуры. Всего определены две такие функции, задаваемые константами GLJMEAREST и GLJJNEAR. По умолчанию используется GLJJNEAR.

GL TEXTURE WRAP S устанавливает параметр сворачивания координаты *s* текстуры в значение GL_CLAMP или GL_REPEAT.

GL_CLAMP фиксирует координату *s* в диапазоне [0, 1], что полезно для предотвращения искусственного сворачивания, когда на объект накладывается один образ.

GL_REPEAT отбрасывает целую часть координаты *s*, т. к. OpenGL использует только значения меньшие 1, и таким образом создается повторяющийся трафарет. Бордюр текстуры доступен только при установке этого параметра в GL_CLAMP. По умолчанию он установлен GL_REPEAT.

GL TEXTURE WRAP_T аналогично GL_TEXTURE_WRAP_S, но только для координаты *t*.

GL_TEXTURE_BORDER_COLOR устанавливает цвет бордюра.

Param определяет значение для параметра *pname*.

GL_NEAREST возвращает значение ближайшего от центра пикселя элемента текстуры.

GL LINEAR возвращает среднеарифметическое значение четырех элементов текстуры, расположенных в центре пикселя.

Линейная фильтрация для четырех ближайших элементов текстуры применима только для двумерного образа. Для одномерного — линейная фильтрация применяется только к двум ближайшим элементам.

Привязка текстуры к объекты осуществляется с помощью текстурных координат.

Пример:

```
glBegin (GL_QUADS);
   glTexCoord2d (1.0, 0.0);
     glVertex3f (-1.0, 1.0, 1.0);
   glTexCoord2d (1.0, 1.0);
```

```
      glVertex3f (1.0, 1.0, 1.0);
    glTexCoord2d (0.0, 1.0);
      glVertex3f (1.0, 1.0,—1.0);
    glTexCoord2d (0.0, 0.0);
      glVertex3f (-1.0, 1.0,—1.0);
glEnd;
```

Лабораторная работа № 7. Интерактивная графика

Цель. Продемонстрировать знание основных возможностей графической библиотеки OpenGL, в том числе работу с системами координат, освещением, текстурированием, спецэффектами (прозрачность, туман).

Задачи. Написать проект, реализующий сложную трехмерную сцену с элементами анимации и интерактивности.

Программа должна обладать следующими возможностями:

- изменять положения наблюдателя и направления наблюдения с течением времени;
- предоставлять пользователю возможности управлять точкой наблюдения;
- перемещать и вращать отдельные объекты сцены (анимация).

Задание предполагает творческий подход. Конкретные элементы сцены, их расположение, взаимодействие и параметры определяются разработчиком исходя из его цели и эстетических соображений.

Методические рекомендации

Ниже приведено несколько примеров заданий, которые могут быть выполнены в качестве данной лабораторной работы. Студент (или бригада студентов) может либо взять одно из этих заданий (по согласованию с преподавателем), либо выбрать свободную тему и реализовать с помощью OpenGL какие-либо свои идеи.

Колебательная система

Написать программу, моделирующую некоторую колебательную систему в трехмерном пространстве. Колебательная система выбирается разработчиком. Программа должна вычислять изменение состояния системы с течением времени и визуализировать систему на экране с учетом вычисленных значений.

Игровая программа (свободная тема)

Написать OpenGL-версию какой-либо игры по выбору разработчика. Игра может быть выбрана из существующих (например, какая-ли-

бо из настольных игр) либо целиком придумана разработчиком. При реализации основное внимание уделить не искусственному интеллекту компьютерных противников (если таковые имеются), а максимальному использованию возможностей OpenGL и удобному пользовательскому интерфейсу.

Демонстрационная программа (свободная тема)

Написать программу, создающую некоторую трехмерную сцену с элементами анимации. В программе должны быть явно продемонстрированы изученные возможности OpenGL по созданию трехмерных изображений (см. общие требования в начале задания).

Примитивы должны формировать изображение, моделирующее какую-либо реальную сцену. Конкретное содержимое сцены, анимация элементов и предоставление пользователю каких-либо возможностей управления определяется разработчиком по своему усмотрению.

Солнечная система

Промоделировать движение планет в некоторой звездной системе. Можно взять за основу как Солнечную систему, так и какую-то произвольно придуманную разработчиком (например, систему с двумя звездами). Орбиты планет для простоты считать окружностями, а сами планеты — сферическими телами. Хотя бы у одной планеты должен быть один или несколько спутников.

В сцене должно использоваться текстурирование (для поверхностей планет), задание разных систем координат относительно друг друга (с использованием видовой матрицы) и освещение. Источником освещения является звезда, к ее свету добавляется некоторое фоновое освещение; кроме того, могут присутствовать дополнительные источники освещения (кометы, спутники и др.).

Морфинг объектов

Целью лабораторной работы является написание программы, которая визуализирует морфинг каких-либо придуманных разработчиком объектов. В общем случае объектов может быть несколько, и морфинг можно осуществлять циклически в определенные интервалы времени. При этом конечный объект можно выбирать как по порядку, так и случайным образом.

Под морфингом понимается процесс плавного преобразования одного геометрического объекта в другой с течением времени. Существует множество различных алгоритмов морфинга, которые обычно

применяются к какому-либо конкретному классу объектов. В данной лабораторной работе предлагается реализовать алгоритм линейного морфинга для объектов, заданных прямоугольной сеткой вершин.

Объекты рассматриваемого типа описываются некоторым двумерным массивом вершин $P[u, v]$, где P является точкой в трехмерном пространстве (возможно, с некоторыми атрибутами: цветом, прозрачностью, нормалью, текстурными координатами и тому подобным). Если два произвольных объекта описаны в такой форме, то в некоторый момент времени t мы можем сформировать описание промежуточного объекта, заданного на той же самой сетке $[u, v]$.

Для простоты будем предполагать, что сетка имеет одинаковые размеры $m \times n$ для всех объектов, а вершины распределены по сетке таким образом, что для каких-либо конкретных координат u, v соответствующие точки объектов находятся в трехмерном пространстве относительно близко. Таким образом, не требуется выполнять вычисление дополнительных вершин (как было бы для сеток разных размеров) и поиск соответствия между вершинами (обычно это поиск наиболее близких точек объектов). Это сильно упрощает реализацию морфинга.

Предполагается, что морфинг начинается в момент времени t_1 и заканчивается в момент t_2.

Промежуточный объект в таком случае может быть вычислен как набор вершин $P_t[u, v]$, полученных путем линейной интерполяции соответствующих вершин начального и конечного объектов:

$$P_t[u, v] = (1-k) \cdot P_1[u, v] + k \cdot P_2[u, v],$$

где k — некоторый коэффициент интерполяции, монотонно возрастающий с течением времени от 0 до 1, $k = f(t)$. В простейшем случае можно принять

$$k = \frac{t - t_1}{t_2 - t_1}.$$

Более сложные зависимости k от t используются для того, чтобы сделать морфинг более плавно начинающимся и заканчивающимся. Например, можно воспользоваться зависимостью $k' = 3k^2 - 2k^3$, где k вычисляется линейно по приведенной выше формуле.

В качестве примеров объектов, заданных на прямоугольной сетке, рассмотрим сферу и цилиндр. Параметры u, v представляют собой целые числа (индексы массива вершин) в диапазоне $[0; m - 1]$ и $[0; n - 1]$ соответственно.

Для сферы радиусом R сетка выглядит следующим образом:

$$x(u,v) = R\cos\alpha\cos\beta;$$
$$y(u,v) = R\sin\alpha\cos\beta;$$
$$z(u,v) = R\sin\beta;$$
$$\alpha = \frac{2\pi u}{m};$$
$$\beta = \frac{\pi v}{n-1} - \frac{\pi}{2}.$$

Для боковой поверхности цилиндра радиусом R и высотой $2R$ сетка выглядит следующим образом:

$$x(u,v) = R\cos\alpha;$$
$$y(u,v) = R\sin\beta;$$
$$z(u,v) = \frac{2Rv}{n-1} - R;$$
$$\alpha = \frac{2\pi u}{m}.$$

Чтобы цилиндр выглядел законченным, можно задать боковую поверхность только в диапазоне $v \in [n/4; 3n/4]$, а на оставшихся двух интервалах создать диски, являющиеся основаниями цилиндра.

Лабиринт

Целью лабораторной работы является создание некоторого игрового приложения на основе OpenGL. В процессе игры главный герой перемещается по лабиринту, стенки которого задаются на некоторой равномерной сетке (рис. 6.2). При этом каждая ячейка сетки либо свободна (есть проход), либо занята стенкой. Фактически для рендеринга такого лабиринта достаточно вывести все занятые ячейки в виде набора из 4 ограничивающих ячейку стенок (хотя это будет неоптимальный метод). Пол и потолок представляют собой две плоскости, расположенные на фиксированной высоте. Обзор окружающего мира производится «из глаз» главного героя, то есть игра относится к жанру игр «от первого лица».

Целью игры может являться сбор всех предметов в лабиринте, поиск выхода или выполнение каких-либо других заданий (по усмотрению разработчика). Помещения могут быть разделены дверями, которые открываются либо автоматически при приближении главного героя, либо каким-то другим образом (заранее собранным ключом,

нажатием кнопки и тому подобным образом). Двери можно представить как ячейки определенного типа в сетке.

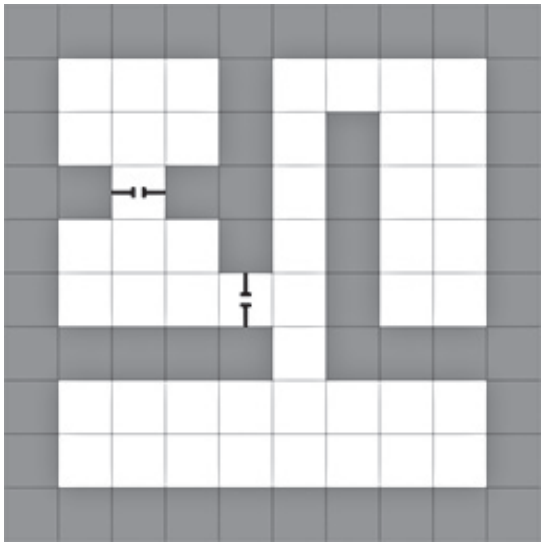

Рис. 6.2. Описание лабиринта на сетке

Управление перемещением в таких играх обычно производится следующим образом:

клавиша W — движение вперед;

S — движение назад;

A — движение влево (без поворота, «ходьба боком»);

D — движение вправо;

«пробел» — прыжок (если такая возможность реализована);

перемещение мыши по горизонтали — поворот головы влево-вправо;

перемещение мыши по вертикали — наклон взгляда вверх-вниз (с ограничением угла наклона);

нажатие кнопки мыши — выполнение какого-либо действия.

Лабиринт (или набор лабиринтов) может загружаться из внешнего файла, ресурсов приложения или просто быть прописан в программе. Рекомендуется размер сетки не менее 30×30.

СЛОВАРЬ СПЕЦИАЛЬНЫХ ТЕРМИНОВ И ПОНЯТИЙ

2D Graphics — двумерная графика, графика на плоскости.

3D Graphics — трехмерная графика. Визуальное отображение трехмерной сцены или объекта. Для представления трехмерной графики на двумерном устройстве применяют рендеринг.

Alpha — коэффициент прозрачности. В описание цвета (RGBA) может входить специальный канал, называемый альфа-каналом (A), который хранит коэффициент прозрачности данного цвета.

Alpha Blending (*Alpha pixel blending*) — реальный мир состоит из прозрачных, полупрозрачных и непрозрачных объектов. *Alpha Blending* — это способ передачи информации о прозрачности полупрозрачным объектам. Эффект прозрачности и просвечивания достигается путем смешивания значений цветов исходного и результирующего пикселей. Разделение изображения на многоугольники производится с использованием маски, плотность которой зависит от прозрачности объекта. В результате цвет точки является комбинацией цветов переднего и заднего плана. Обычно Alpha имеет нормализованное значение от 0 до 1 для каждого пикселя.

Alpha Buffer — альфа-буфер. Дополнительный буфер (память), в котором содержится информация о прозрачности, таким образом, пиксель имеет четырехзначное представление (RGBA), и в 32-разрядном буфере содержится 24 бита информации о цвете, т. е. 8 битов на каждый из цветов (красный, зеленый и синий) и 8 битов на значение alpha.

Ambient — световой источник, который светит одинаково во всех направлениях. Все объекты освещаются с равной интенсивностью.

Atmospheric Effect — специальные эффекты, например, туман, позволяющие улучшить рендеринг изображений реального мира.

BMP — формат хранения растровых изображений. Глубина цвета в данном формате может быть 1, 2, 4, 8, 16, 24, 32, 48 битов на пиксель. В формате BMP-изображения могут храниться как есть или же с применением некоторых распространенных алгоритмов сжатия. В частности, формат BMP поддерживает RLE-сжатие без потери качества.

Bump Mapping — техника симуляции неровностей на плоской поверхности без больших вычислительных затрат и изменения геометрии. Для каждого пикселя поверхности выполняется вычисление освещения на основе значений в специальной карте высот, называемой *bump map*. Обычно это 8-битная черно-белая текстура, и значения цвета текстуры не накладываются, как обычные текстуры, а используются для описания неровности поверхности. Цвет каждого текселя определяет высоту соответствующей точки рельефа, большие значения означают большую высоту над исходной поверхностью, а меньшие — меньшую. Для моделирования неровностей для разных точек полигона задаются нормали к поверхности, которые учитываются при вычислении попиксельного освещения. В результате получается более натуральное изображение поверхности.

Colored lighting — цветовое освещение. Освещение источниками разного цвета, при этом происходит смешение цветов.

Depth Cueing — уменьшение интенсивности освещения текстур при удалении объекта от точки наблюдения.

Directional — световой источник, который освещает одинаково все объекты сцены, как бы из бесконечности в определенном направлении. Обычно используется для создания удаленных световых источников (таких как солнце).

Displacement Mapping — метод наложение карт путем добавления деталей к трехмерным объектам. Карты смещения позволяют получить настоящие сложные 3D-объекты из вершин и полигонов без ограничений, присущих попиксельным методам. Этот метод изменяет положение вершин треугольников, сдвигая их по нормали на величину, вычисленную из значений в картах смещения. Карта смещения (*displacement map*) — это обычно черно-белая текстура, и значения

в ней используются для определения высоты каждой точки поверхности объекта (значения могут храниться как 8-битные или 16-битные числа), что схоже с bumpmap. Часто карты смещения используются (в этом случае они называются и картами высот) для создания земной поверхности с холмами и впадинами. Поскольку рельеф местности описывается двухмерной картой смещения, его относительно легко деформировать при необходимости, так как это потребует всего лишь модификации карты смещения и рендеринга на ее основе поверхности в следующем кадре.

Flat Shading (*Flat*) — метод затенения, называемый также постоянным затенением. Поверхность объекта, построенного с использованием этого метода, получается наиболее низкого качества, и изображение выглядит как бы поделенным на блоки.

Gouraud Shading (*Smooth shading*) — затенение методом Гуро (или плавное затенение) — один из наиболее популярных алгоритмов затенения, который обеспечивает прорисовку плавных теней вокруг изображаемого объекта, что позволяет изображать трехмерные объекты на плоском экране.

Normal Mapping — улучшенная разновидность техники расчета нормалей при выводе поверхности изображения. Полностью заменяет нормали при помощи выборки их значений из специально подготовленной карты нормалей (*normal map*). Эти карты обычно являются текстурами с сохраненными в них заранее просчитанными значениями нормалей, представленными в виде компонент цвета RGB.

Occlusion — эффект перекрытия в трехмерном пространстве одного объекта другим.

Palletized Texture — формат хранения текстур в сжатом виде (1-, 2-, 4-и 8-битный формат вместо 24-битного). Обеспечивает возможность хранения большего числа текстур в меньшем объеме памяти.

Parallax Mapping — метод наложения текстуры на поверхность, при котором отображается большее количество деталей поверхности, чем есть в исходной геометрической модели. *Parallax Mapping* является техникой аппроксимации эффекта смещения точек поверхности в зависимости от изменения точки зрения. Техника сдвигает текстурные координаты так, чтобы поверхность выглядела более объемной. Идея метода состоит в том, чтобы возвращать текстурные координаты той точки, где видовой вектор пересекает поверхность. Это требует просчета лучей (рейтрейсинг) для карты высот, но если она не имеет слишком

сильно изменяющихся значений («гладкая» или «плавная»), то можно обойтись аппроксимацией. Такой метод хорош для поверхностей с плавно изменяющимися высотами, без просчета пересечений и больших значений смещения.

Parallel point — световой источник, который освещает равномерно все объекты параллельным пучком света.

Perspective Correction — один из способов создания реалистичных объектов. Рассматриваются величины *Z* (глубина) при разделении объекта на многоугольники. При создании современных игр разработчики обычно используют довольно большого размера треугольники для описания поверхности объекта, а также текстурные карты для более точного и детального изображения. Если 3D-объект движется от наблюдателя, то уменьшаются его линейные размеры (высота и ширина). Без использования функции *perspective correction* объект будет дергаться и двигаться нереалистично. С каждым уровнем скорректированной перспективы происходят изменения на пиксель в зависимости от глубины.

Phong Shading — наиболее эффективный из всех известных методов затенения, позволяющий получить реалистичное освещение. Реалистичность достигается за счет вычисления объема освещения для каждой точки вместо множества многоугольников. Каждый пиксель получает свой собственный цвет на основе модели освещения, направленного на этот пиксель.

RGB — система цветообразования, в которой конечный цвет получается за счет смешения с различной интенсивностью трех основных цветов: красного (Red), зеленого (Green) и синего (Blue).

Spot — световой источник, ограниченный пространством конуса. Он светит не во всех направлениях, а в пределах некого конуса. Освещаются только объекты, попадающие в этот конус.

Stippling — процесс создания каркасных изображений.

Texture Anti-aliasing — удаление нежелательных искажений растровых изображений с помощью интерполяции текстурных изображений.

Transformation (изменение координат) — последовательность математических операций над выходными графическими примитивами и геометрическими атрибутами для преобразования их из расчетных координат в системные координаты.

Vertex — точка в трехмерном пространстве, заданная координатами.

Z-buffer — часть графической памяти (буфер), в которой хранятся

расстояния от точки наблюдения до каждого пикселя (значения *Z*). *Z-buffer* определяет, какая из многих перекрывающихся точек наиболее близка к плоскости наблюдения. Обычно *z*-буфер имеет не менее 16 битов на пиксель для представления глубины цвета. Аппаратные акселераторы 3D-графики могут иметь собственный *z*-буфер на графической карте, чтобы избежать удвоенной нагрузки на системную шину при передаче данных. Некоторые реализации *Z*-buffer используются для хранения не целочисленного значения глубины, а значения с плавающей запятой от 0 до 1.

Z-buffering — процесс удаления скрытых поверхностей, использующий значения глубины, хранящиеся в *Z*-буфере. Перед отображением нового кадра буфер очищается, и значения величин *Z* устанавливаются равными дальней плоскости отсечения. При рендеринге объекта устанавливаются значения *Z* для каждого пикселя: чем ближе расположен пиксель, тем меньше значение величины *Z*. Для каждого нового пикселя значение глубины сравнивается со значением, хранящимся в буфере, и пиксель записывается в кадр только в том случае, если величина глубины меньше сохраненного значения.

Альфа-канал (*alpha channel*) — дополнительный канал растровых данных, используемый для хранения сведений о прозрачности изображения. Степень прозрачности пикселя, заданная восьмибитовым альфа-значением, находится в интервале от 0 (пиксель полностью невидим — прозрачен) до 255 (пиксель полностью виден — непрозрачен).

Антиалиасинг (*anti-aliasing*) — способ обработки (интерполяции) пикселей для получения более четких краев (границ) изображения (объекта). Наиболее часто используемая техника для создания плавного перехода от цвета линии или края к цвету фона.

Бинаризация — это преобразование изображения, в общем случае к одноцветному (чаще всего к черно-белому). При этом выбирается некий порог (например, 128), все значения ниже которого превращаются в цвет фона, а выше — в основной цвет.

Буфер — область временного хранения данных, часто используется для компенсации разницы в скорости работы различных компонентов системы.

Буфер глубины (*Z-buffer*, *depth buffer*) — двумерный массив данных, дополняющий двумерное изображение, где для каждого пикселя (фрагмента) изображения сопоставляется «глубина» (расстояние от наблюдателя до поверхности изображаемого объекта). *Z*-буфер представля-

ет собой двумерный массив, каждый элемент которого соответствует пикселю на экране. Когда видеокарта рисует пиксель, его удаленность просчитывается и записывается в ячейку *z*-буфера. Если пиксели двух объектов перекрываются, то их значения глубины сравниваются и выводится на экран тот, который ближе, а его значение удаленности сохраняется в буфер.

Буфер кадра (*Frame buffer*) — специально отведенная область памяти компьютера или отдельной платы для временного хранения данных о пикселях, требуемых для отображения одного кадра (полного изображения) на экране монитора. Емкость буфера кадра определяется количеством битов, задействованных для определения каждого пикселя, который должен отображать изменяемую область или количество цветов и их интенсивность на экране.

Векторная графика — способ представления объектов и изображений в компьютерной графике, основанный на использовании элементарных геометрических объектов, таких как точки, линии, сплайны и многоугольники. Объекты векторной графики являются графическими изображениями математических функций.

Вершинный шейдер (*Vertex Shader*) — оперирует данными, сопоставленными с вершинами многогранников. К таким данным, в частности, относятся координаты вершины в пространстве, текстурные координаты, тангенс-вектор, вектор бинормали, вектор нормали. Вершинный шейдер может быть использован для видового и перспективного преобразования вершин, генерации текстурных координат, расчета освещения и т. д.

Вторичный буфер (*Back buffer*) — область памяти, в которой рассчитываются объекты трехмерной сцены. Вывод изображения на экран осуществляется через *Front Buffer* (первичный буфер). Таким образом достигается плавная смена кадров.

Глубина цвета — термин компьютерной графики, означающий объем памяти в количестве битов, используемых для хранения и представления цвета при кодировании одного пикселя растровой графики или видеоизображения. Выражается единицей бит на пиксель.

Графический редактор — программное средство для создания и обработки изображений.

Интерполяция (*Interpolation*) — математический способ восстановления отсутствующей информации. Например, необходимо увеличить

размер изображения в 2 раза, со 100 пикселей до 200. Недостающие пиксели генерируются с помощью интерполяции пикселей, соседних с тем, который необходимо восстановить. После восстановления всех недостающих пикселей получается 200 пикселей вместо 100 существовавших, и, таким образом, изображение увеличилось вдвое.

Конвейер — серия шагов по созданию и отображению трехмерного изображения. Первый шаг — трансформация: создается трехмерный объект и отображается на плоскость. Второй шаг — добавление освещенности объекту. Третий шаг — рендеринг цветов и теней многоугольников для соответствующих текстур.

Контраст — разность максимального и минимального значений яркости.

Пиксель (сокращение от слов *picture cell* — элемент изображения) — термин, обозначающий элемент изображения, который является наименьшим элементом экрана монитора.

Пиксельный шейдер (*Pixel Shader*) — работает с фрагментами изображения. Под фрагментом изображения в данном случае понимается пиксель, которому поставлен в соответствие некоторый набор атрибутов, таких как цвет, глубина, текстурные координаты. Фрагментный шейдер используется на последней стадии графического конвейера для формирования фрагмента изображения.

Разрешение — величина, определяющая количество точек (элементов растрового изображения) на единицу площади (или единицу длины).

Растеризация (*Rasterization*) — процесс преобразования графического векторного изображения в растровый вид.

Растровое изображение — представляет собой сетку пикселей или цветных точек (обычно прямоугольную) на компьютерном мониторе, бумаге и других отображающих устройствах и материалах. Важными характеристиками изображения являются: количество пикселей, глубина цвета, цветовое пространство, разрешение.

Режим реального времени (*Real-time*) — процесс, в котором имитируемые события происходят так же, как и в реальной жизни.

Рендеринг — процесс получения изображения по модели с помощью компьютерной программы. Здесь модель — это описание любых объектов или явлений на строго определенном языке или в виде структуры данных. Такое описание может содержать геометрические данные, положение точки наблюдателя, информацию об освещении, сте-

пени наличия какого-то вещества, напряженность физического поля и пр.

Тексель — минимальная единица текстуры трехмерного объекта. Пиксель текстуры. Текстура, в свою очередь, представляет собой массив текселей.

Текстура — 1. Растровое изображение, накладываемое на поверхность полигона, из которых состоят 3D-модели, для придания ей цвета, окраски или иллюзии рельефа. 2. Двумерное изображение, хранящееся в памяти компьютера (CPU) или графического адаптера (GPU) в одном из пиксельных форматов.

Тесселяция (*Tessellation*) — процесс деления изображения на более мелкие формы. Для описания характера поверхности объекта она делится на всевозможные многоугольники. Наиболее часто при отображении графических объектов используется деление на треугольники, поскольку только треугольники позволяют однозначно определить ориентацию плоскости в пространстве.

Трассировка лучей (*Ray Tracing*) — один из самых сложных и качественных методов построения реалистических изображений. Наиболее распространен вариант «обратной трассировки лучей»: от глаза наблюдателя, через пиксель строящегося изображения, проводят луч и, учитывая все его отражения от объектов, вычисляют цвет этого пикселя.

Трехмерная графика — совокупность приемов и инструментов (как программных, так и аппаратных), предназначенных для изображения объемных объектов. Для создания модели трехмерного объекта используются геометрические примитивы (куб, параллелепипед, шар, эллипсоид, конус и др.) и гладкие поверхности, описываемые кусочно-гладкими бикубическими полиномами. Вид поверхности задается сеткой расположенных в пространстве опорных точек. Участки поверхности между опорными точками — границы объекта, которые обладают различными свойствами и могут быть гладкими, шероховатыми, прозрачными, непрозрачными, зеркальными и т. п. В соответствии с этими свойствами поверхности закрашиваются тем или иным способом. Движение объектов и анимация воспроизводятся суперпозицией аффинных преобразований.

Цветовое пространство — модель представления цвета, основанная на использовании цветовых координат. Виды цветовых пространств: RGB, CMYK, CHL, CHB, Lab и др.

Шейдер — программа для одной из ступеней графического конвейера, используемая в трехмерной графике для определения окончательных параметров объекта или изображения. Шейдеры делятся на два типа: вершинные (геометрические) и фрагментные (пиксельные).

Яркость цифрового изображения — величина уровней интенсивности в пиксельной матрице изображения, снятого цифровой камерой или оцифрованного аналого-цифровым преобразователем. Яркость — это величина уровней интенсивности всех пикселей вместе, составляющих цифровое изображение, которое было снято, оцифровано и отображено на экране.

РЕКОМЕНДУЕМЫЙ БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Боресков А. В. Графика трехмерной компьютерной игры на основе OpenGL / А. В. Боресков. М. : Диалог–МИФИ, 2004. 384 с.

2 OpenGL. Официальное руководство программиста : пер. с англ. / Ву Мейсон [и др.]. СПб. : ДиаСофтЮП, 2002. 592 с.

3 Желтов В. С. Моделирование осветительных установок на основе решения уравнения глобального освещения локальными оценками метода Монте-Карло : дис. ... канд. техн. наук : 05.09.07 / Желтов Виктор Сергеевич ; [Моск. энергет. ин-т]. М., 2008. 94 с.

4 Компьютерная графика. Уроки, алгоритмы, программы, примеры [Электрон. ресурс]. Режим доступа: http://grafika.me/. Загл. с экрана.

5 OpenGL. Руководство по программированию / М. Ву [и др.]. СПб. : Питер, 2006. 624 с.

6 Райт Ричард С. OpenGL. Суперкнига / Ричард С. Райт, мл. Бенджамин Липчак. 3-е издание. СПб. : Вильямс, 2006. 1040 с.

7 Роджерс Д. Математические основы машинной графики / Д. Роджерс, Дж. Адамс. М. : Мир, 2001. 604 с.

8 Рост Рэнди Дж. OpenGL. Трехмерная графика и язык программирования шейдеров. Для профессионалов / Дж. Рост Рэнди. СПб. : Питер. 2005. 432 с.

9 Строзотт Т. Нефотореалистичная компьютерная графика: моделирование, рендеринг, анимация / Т. Строзотт, Ш. Шлехтвег. М. : КУДИЦ-ОБРАЗ, 2005. 416 с.

10 Хилл Ф. OpenGL. Программирование компьютерной графики. Для профессионалов / Ф. Хилл. Спб. : Питер, 2002. 1088 с.

11 Эдвард Эйнджел. Интерактивная компьютерная графика. Вводный курс на базе OpenGL : пер. с англ. / Эйнджел Эдвард. М. : Вильямс, 2001. 592 с.

Учебное издание

Папуловская Наталья Владимировна

МАТЕМАТИЧЕСКИЕ ОСНОВЫ ПРОГРАММИРОВАНИЯ ТРЕХМЕРНОЙ ГРАФИКИ

Редактор *И. В. Меркурьева*
Верстка *Е. В. Ровнушкиной*

Подписано в печать 15.12.2016. Формат 70×100 1/16.
Бумага писчая. Цифровая печать. Усл. печ. л. 9,03.
Уч.-изд. л. 5,8. Тираж 100 экз. Заказ 362.

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: 8 (343) 375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620075, Екатеринбург, ул. Тургенева, 4
Тел.: 8 (343) 350-56-64, 350-90-13
Факс: 8 (343) 358-93-06
E-mail: press-urfu@mail.ru

ПАПУЛОВСКАЯ НАТАЛЬЯ ВЛАДИМИРОВНА

Кандидат педагогических наук, доцент, преподаватель
Уральского федерального университета